Diplomarbeit

Sichere und verlässliche Kommunikation zwischen Low-Cost-Devices und PCs

Patric de Waha
Mai 2007

Betreuer: Prof. Dr. Paul Müller
Dipl.-Ing. Dirk Henrici

Fachbereich Informatik

AG Integrierte Kommunikationssysteme
Universität Kaiserslautern • Postfach 3049 • 67653 Kaiserslautern

Erklärung

Hiermit erkläre ich, dass ich die vorgelegte Arbeit
" Sichere und verlässliche Kommunikation zwischen Low-Cost-Devices und PCs ",
selbständig und ohne fremde Hilfe verfasst und keine anderen als die
angegebenen Hilfsmittel verwendet habe.

Kaiserslautern, 14.05.2007

Abstract

Die zunehmende Präsenz intelligenter elektronischer Geräte und vor allem deren Vernetzung stellt uns vor einige neue Herausforderungen. Sicherheit und Kosten sind nur einige der ausschlaggebenden Probleme mit denen wir uns konfrontiert sehen. Das Anbinden von einfachen Geräten an große Netze, vor allem an das Internet, wirft Probleme auf, die Zuwendung bedürfen. So sind z.B. Kosten und Sicherheit eines der wesentlichen Faktoren, die in diesen Szenarien genauerer Untersuchung bedürfen. In dieser Diplomarbeit, werden Aspekte wie Ausfallsicherheit, Kosten und sichere Kommunikation im Kontext des "Ubiquitous computing" besprochen. Anschließend werden Konzepte vorgeschlagen, die diesen Problemen Rechnung tragen. Die Funktionsfähigkeit der erarbeiteten Methoden und Konzepte, wird anhand eines realen Fallbeispiels und eines Demonstrator, nachgewiesen.

Inhaltsverzeichnis

EINLEITUNG **7**

AUSGANGSLAGE 7

ZIELE DER DIPLOMARBEIT 8

AUFBAU DER DIPLOMARBEIT 9

PROBLEMSTELLUNG **10**

TECHNISCHE ASPEKTE 12

Hohe Verfügbarkeit *15*

Geringe Kosten *18*

Flexibilität und Erweiterbarkeit *19*

Schnittstelle zum Internet *19*

ZUSAMMENFASSUNG 20

ARCHITEKTUR **21**

KNOTEN 21

Verwaltung *22*

Vernetzung *22*

DIE PEER-APPLIKATION 23

VERNETZUNG VON PEER UND GERÄTE-CONTROLLER 25

KLIENT-APPLIKATION 25

SERVICE-ARCHITEKTUR 25

GESAMTARCHITEKTUR 26

ANDERE LÖSUNGSKONZEPTE 27

SCADA *27*

WSN *28*

INSIGHT *29*

INTERNET0 ..... *30*

GSN *31*

SASI *31*

TECHNOLOGIEWAHL **33**

KNOTEN 33

CAN-Bus *34*

I2C-Bus *34*

1W-Bus ... *35*
Vergleich der Busse ... *35*
GERÄTE-CONTROLLER UND VERNETZUNG ... 36
RCM4000 RabbitCore ... *37*
MAXIM-IC ... *37*
Vergleich der Mikrocontroller ... *38*
DIE PEER-PLATTFORM ... 38
ZUSAMMENFASSUNG ... 39
ENTWURF DER KOMMUNIKATION ... 40
KOMMUNIKATION ZWISCHEN GERÄTE-CONTROLLER UND KNOTEN ... 41
PROTOKOLL ZWISCHEN GERÄTE-CONTROLLER UND PEER-APPLIKATION ... 42
Anforderungen an das Kommunikationsprotokoll ... *44*
Datenaustausch zwischen Geräte-Controller und Peer ... *47*
Zusammenfassung ... *48*
PEER-APPLIKATION UND KLIENT-APPLIKATION ... 49
Schnittstellenauswahl ... *50*
INFORMATIONSBASIS ... 51
ENTWURF DER SOFTWARE ... 54
GERÄTE-CONTROLLER SOFTWARE ... 54
PEER-APPLIKATION ... 58
EINSATZ ... 62
SZENARIO ... 62
AUFBAU ... 63
VERSUCH ... 65
ZUSAMMENFASSUNG ... 67
AUSBLICK ... 69
ANHANG ... 70
BENUTZER– UND ENTWICKLERHANDBUCH ... 70
SASI ... *70*
Architektur ... *71*
Installation und Konfiguration ... *73*
Weiterentwicklung ... *76*
Knoten ... *77*

Geräte-Controller *77*
Peer-Applikation *78*
SOFTWAREENTWÜRFE 81
Geräte-Controller Software *81*
Peer Software *82*
Paket-Logik des Geräte-Controller *83*
Paket-Logik der Peer-Applikation *84*
LITERATURVERZEICHNIS 85
ABBILDUNGSVERZEICHNIS 87
ABKÜRZUNGEN 88

Einleitung

Ausgangslage

Die zunehmende Durchdringung der Informatik im Alltag und in unserem Umfeld wird auch in Zukunft noch zunehmen. Mit modernen Chips ist schon heute mehr Rechenleistung in einem 10 Euro teuren Mikrocontroller als in einem PC vor zehn Jahren. Dies hat zur Folge, dass immer mehr alltägliche Gegenstände mit Rechenleistung ausgestattet werden. Sei es, um mehr Funktionalität zu erreichen oder um die Nutzbarkeit der Gegenstände zu verbessern. Die Gründe für die Durchdringung der Informatik im Alltag sind vielseitig. Die Vision des ***„Ubiquitous computing"*** ist bereits heute teilweise zur Realität geworden. Kraftfahrzeuge, die bereits einige Kilometer Kabel enthalten, die mit leistungsfähigen Rechnern verbunden sind, sind nur die Spitze des Eisbergs. Diese scheinbar unvermeidbare Entwicklung wirft aber auch einige Fragen auf. Wie garantieren wir Sicherheit vor Ausfall oder Missbrauch durch andere? Diese unerwünschten Konsequenzen, die mit ***„Ubiquitous computing"*** verbunden sind, potenzieren, sich indem man diese intelligenten Gegenstände vernetzt. Die Vernetzung dieser Gegenstände ist ein genauso logischer Schritt wie die Ausstattung der Gegenstände mit Chips und birgt mindestens genauso viele Probleme. Wie kann man sichere Kommunikation mit diesen Gegenständen garantieren? Insbesondere wenn die Gegenstände mit großen Netzen wie dem Internet verbunden sind.

Da diese Entwicklung weiter voranschreitet, stellt sich nicht zuletzt die Frage der Kosten. Wenn einfache Gebrauchsgegenstände mit Mikroprozessoren ausgestattet werden und diese in großer Zahl in unserem Umfeld vorhanden sind, kann man davon ausgehen, dass bedingt durch diese elektronischen Komponenten die Kosten für die Nutzer steigen. Kann man Konzepte erarbeiten, die diese Kosten minimieren? Gibt es Architekturen, die in diesem Zusammenhang besonders vorteilhaft sind?

Mit einigen dieser Fragen wird sich in dieser Diplomarbeit auseinandergesetzt. Das Feld des „***Ubiquitous computing***" bedient viele Bereiche der Wissenschaft. Von gesellschaftlichen Fragen bis hin zu wirtschaftlichen und nicht zuletzt informationstechnischen Fragen sind noch einige offen.

Ziele der Diplomarbeit

Im vorherigen Abschnitt wurden einige Probleme herausgearbeitet, die mit dem ***„Ubiquitous computing"*** verbunden sind. Die identifizierten Probleme sollen zum Anlass genommen werden, bessere Architekturen und Konzepte, die diese Probleme lösen könnten, auszuarbeiten. Der Schwerpunkt liegt in der Konzeption eines Rahmenwerks, das von den einfachen elektronischen Gegenständen bis hin zu Internet-Applikationen eine Brücke schlägt. Dazu ist es nötig, die Anforderungen zu ermitteln und zu bewerten. Das Ziel ist es, eine Konzeption zu erstellen, die die identifizierten Anforderungen erfüllt. Die Kommunikation zwischen den einzelnen Komponenten sowie die Software selbst, die zum Einsatz kommt, werden in dieser Diplomarbeit entwickelt und implementiert. Im Folgenden wird der Begriff **SASI** verwendet, um auf das zu entwickelnde Rahmenwerk zu beziehen. SASI steht für: "Secure Attachment of Sensors to the Internet"

Da sich diese Diplomarbeit mit der praktischen Anwendung von ***„Ubiquitous computing"*** beschäftigt, sollen die Resultate dieser Arbeit auch in einem praktischen Anwendungsbeispiel aufgezeigt werden. Dies impliziert, dass geeignete technische Mittel für den Demonstrator ausgewählt werden. Auf den ausgewählten Komponenten werden die Konzepte implementiert und schlussendlich demonstriert.

Gegen Ende wird auf prinzipielle Vor- und Nachteile der erarbeiteten Konzepte eingegangen. Abgeschlossen wird mit einem Ausblick auf sinnvolle Erweiterungen des Rahmenwerks.

Aufbau der Diplomarbeit

Die Aufgabenstellung dieser Diplomarbeit gliedert sich folgende Teilgebiete:

- Erfassung und Analyse der Problemstellung
- Konzeption und Analyse einer geeigneten Lösung
- Wahl der technischen Hilfsmittel zur Realisierung der Lösung
- Entwurf und Implementieren dieser Lösung
- Erfolg am Beispiel eines Demonstrators nachweisen

In jedem Kapitel wird eine Übersicht angeboten, bevor sich den eigentlichen Problemen zugewandt wird. Jedes Kapitel versucht Kriterien herauszuarbeiten, die die Erkenntnisse des jeweiligen Kapitels festhalten. Diese festgelegten und explizit angegebenen Kriterien gelten als Richtlinien für die weitere Verfeinerung im jeweils darauf folgenden Kapitel.

Problemstellung

Die in der Einleitung angesprochene Entwicklung des ***„Ubiquitous computing"*** und die damit eingeführten Probleme werden in diesem Teil der Diplomarbeit etwas ausführlicher behandelt.

Bereits heute sind eine Vielzahl von alltäglichen Gegenständen mit Mikroprozessoren ausgerüstet, die die Geräte effektiver oder sogar billiger machen sollen. Früher durch elektronische Schaltungen geregelte Apparaturen wurden mit der Zeit mit ICs optimiert, denn weniger Bauteile bedeutet weniger Kosten. Solange Moores Gesetz noch Gültigkeit besitzt, wird diese Entwicklungsgeschwindigkeit uns weiter mit Hochleistungschips, die immer kleiner und billiger werden, versorgen. So z.B. überschreitet die Rechenleistung eines modernen Handys die Leistung eines Rechners der 90er Jahre um Weiten.

Diese Entwicklung führt zu immer neuen Applikationsfeldern der Informatik. Für Geräte, die bisweilen unvernetzt und elektronikfrei ihren Dienst verrichtet haben, ergeben sich nun neue Möglichkeiten. So können z.B. durch Vernetzung der Gegenstände, neue Gebrauchsmuster entstehen. Das Fernsteuern oder intelligente Regeln von bestimmten Apparaturen wären ohne diese Entwicklung nicht möglich. Diese Entwicklung wurde bereits von Marc Weiser im Jahr 1988 erahnt [WEISER] und ist heute bereits zum Teil Realität. Der von Weiser geprägte Begriff des „***Ubiquitous computing***" schildert genau die Entwicklung, die sich seit einigen Jahren anzudeuten scheint, nämlich: Das Durchdringen der Informatik in unserem Alltag, auf transparente und unscheinbare Art und Weise.

Dieser Sachverhalt ist Grund genug, sich mit einigen Aspekten des „***Ubiquitous computing***" zu beschäftigen. Im Rahmen dieser Diplomarbeit stehen die technischen Problemstellungen im Vordergrund, soziale oder wirtschaftliche Fragestellungen werden in diesem Dokument nicht behandelt.

Neben klassischen Anforderungen der Informatik, ergeben sich auch ganz neue Aspekte, die erst in den letzten Jahren an Bedeutung gewonnen haben. Auf diese klassischen und neuen Aspekte soll im weiteren kurz eingegangen werden.

Laut Marc Weiser [WEISER] sollte diese Allgegenwart der Informatik transparent für den Nutzer sein. Der Nutzer sollte sich nicht darüber bewusst sein, ob oder wie die Informatik in dem zu benutzenden Gegenstand zum Tragen kommt. Dies führt unweigerlich dazu, dass eine Ausfallzeit von Null erreicht werden sollte. Der durchschnittliche Nutzer, der nicht über IT-Kenntnisse verfügt, sollte nicht mit technischen Problemen des Gegenstandes in Berührung kommen. Die Ausfallsicherheit kommt besonders dann zum Tragen, wenn die Gegenstände vernetzt werden.

Aktueller den je ist das Thema der Privatsphäre und Vermeidung von unerlaubtem Zugriff auf vertrauliche Daten von privaten Anwendern. In Zeiten in denen Sicherheitslücken in Rechnern dazu genutzt werden um unerlaubt Malware einzuschleusen, stellt sich die Frage ob die Vernetzung von Gegenständen vor allem mit dem Internet, diesem Problem nicht noch weiteren Nährboden liefert. Kann es dazu kommen, dass Personen unerlaubt Geräte beeinflussen können? Können fremde Personen an vertrauliche Informationen gelangen?

Nicht zuletzt spielen in diesem Bereich die Kosten eine wichtige Rolle. Inwieweit können alltägliche Gegenstände zu welchem Preis intelligenter gemacht werden? Es bedarf Konzepte, die diese Entwicklung zu vergleichsweise geringen Kosten ermöglichen. Auch wenn die Einzelkosten eines Mikrochips sehr gering sind, so können höhere Einzelpreise in der Summe doch erheblich sein.

Technische Aspekte

Im Folgenden wird auf konkrete technische Aspekte, die in dieser Diplomarbeit als wichtig erachtet werden, eingegangen. Konzentriert wird sich auf die Ausfallsicherheit, Kommunikationssicherheit und Kostenreduktion.

In diesem Zusammenhang soll kurz auf den aktuellen Stand der Dinge eingegangen werden. Wie bereits erwähnt, existieren eine große Zahl von kostengünstigen Mikrochips, die einem ermöglichen, Gegenstände mit "Intelligenz" auszustatten und zu vernetzen. Besonders spezialisierte Mikrocontroller bieten sogar im ROM integrierte IP-Stacks an. Somit ist es einfach möglich, ein Gerät IP-fähig zu machen. Es ist durchaus üblich, einfache Geräte wie Webcams IP-fähig zu machen und zusätzlich noch einen integrierten Web-Server anzubieten. In dieser Diplomarbeit werden im weiteren Verlauf, Sensoren, Aktuatoren oder auch höherwertigere Komponenten wie z.B. Webcams, als Knoten bezeichnet.

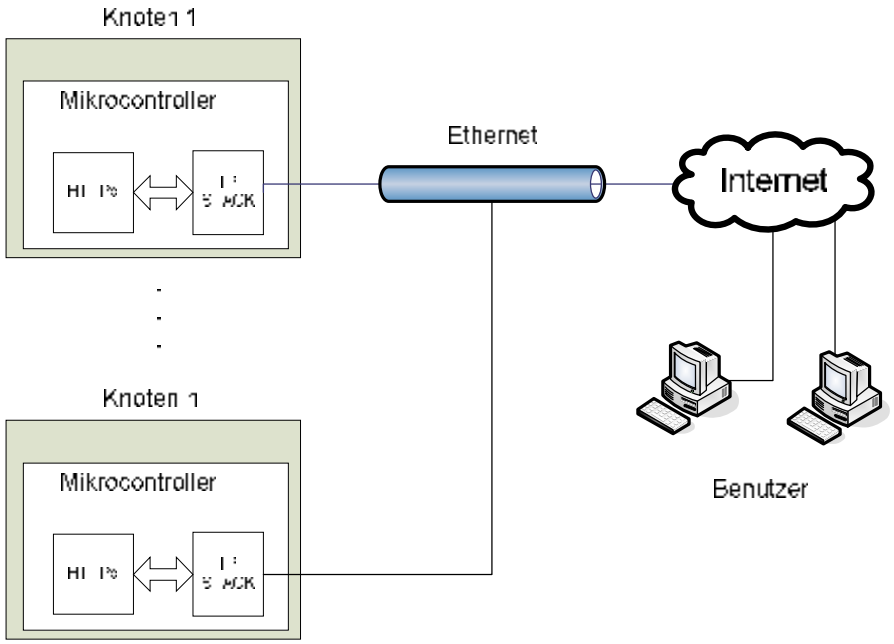


Abbildung 1 IP-fähige Knoten

Das gleiche findet man bei IP-RPS[1] und vielen anderen Geräten, siehe Abbildung 1. Die alternative und auch ältere Methode besteht darin, einen PC als Gateway zu nutzen. Man verbindet die Geräte über eine gegebene Schnittstelle mit dem PC. Die mitgelieferte Software übernimmt dann die Bereitstellung der Daten. Verbreitete Schnittstellen für diesen Zweck sind z.B. USB, FireWire oder die klassische serielle Schnittselle. Die beiden genannten Methoden haben Vor- und Nachteile, auf die im Folgenden eingegangen wird.

Bei einigen kommerziell verfügbaren Sensornetzwerken ist es durchaus üblich, einen kompletten PC als Gateway zu nutzen. Die Sensordaten werden per USB eingelesen und über die kommerzielle Software dem Internet angeboten, siehe Abbildung 2. In Zusammenhang mit dem *„Ubiquitous computing"* jedoch ergeben sich ungewollte Merkmale. Das Einrichten eines zentralen Rechners widerspricht dem Aspekt der transparenten Präsenz der Informationstechnologie. Weiterhin sind die Kosten für einen kompletten Rechner ganz und gar nicht verschwindend gering. Eine Lösung, die Sensornetzwerke mit großen Netzwerken wie dem Internet verbindet und dabei auf PC-Systeme verzichtet, ist vorteilhaft.

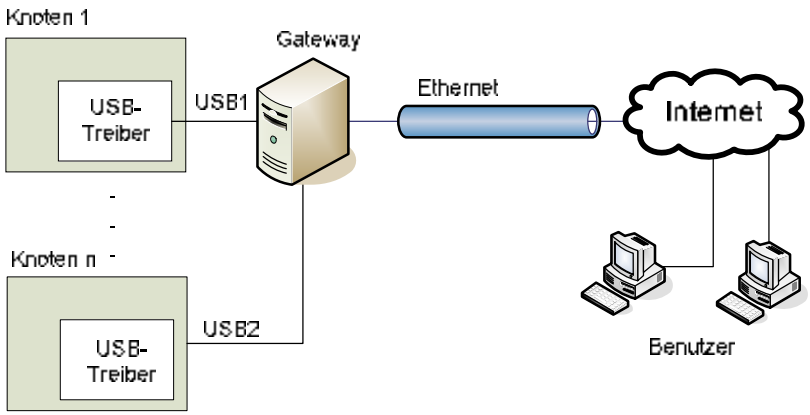


Abbildung 2 Geräteanschluss über Gateway

Den Verzicht auf ein PC-System in der Kette vom Sensor bis hin zum Internet erreicht man durchaus auch mit aktuell verfügbaren Mitteln. Die Mikrochipindustrie ist dem Wunsch nach „Embedded Internet Computing" nachgekommen und hat eine große Anzahl von solchen

[1] Remote Power Switch

Mikrochips entwickelt. Die große Anzahl von Mikrocontrollern auf dem Markt, die bereits über Ethernet-Schnittstellen verfügen, vereinfachen die Integration der Informationstechnik in alltägliche Gegenstände. Integrierte IP-Stacks sind auch keine Seltenheit mehr. Diese Entwicklung, die prinzipiell dem entspricht, was man sich für ***„Ubiquitous computing“*** erwartet, besitzt auch einige Nachteile. Besonders im Bereich der Ausfall- und Informationssicherheit sind solche eingebettete internetfähige Mikrochips sehr anfällig.

Die schwache Ausfallsicherheit bei internetfähigen Mikrocontrollern ist mit einer positiven Eigenschaft dergleichen verbunden. Mit der Größe und auch relativ beschränkten Rechenleistung gehen wenig Kosten einher. Diese beschränkte Rechenleistung kann aber dazu ausgenutzt werden, um einen Mikrocontroller gezielt zu überlasten. Allein die Komplexität und Anzahl der Protokolle bedeuten einen nicht unwesentlichen Anteil an Rechenzeit für einen Mikrocontroller. Es ist durchaus möglich, einen integrierten Webserver ohne Absicht zu überlasten. Bei gleichzeitigem Zugriff mehrerer Nutzer können schnell die Ressourcen aufgebraucht werden und das Gerät kann nicht mehr zuverlässig genutzt werden. Prinzipiell ist die Skalierbarkeit dieser Mikrocontroller doch recht schwach. Auch dies wird der Anforderung nach hoher Verfügbarkeit nicht gerecht. Verfügbarkeit ist ein wichtiges Merkmal, das jedoch in diesem Fall nur beschränkt gewährleistet werden kann.

Ein weiteres Argument, das gegen die Konzeption des „Embedded Internet Computing“ spricht, ist die der dezentralen Speicherung von Authentifikationsdaten. Bei mehreren Geräten, die alle verschiedene Zugangsmechanismen oder verschiedene Authentifikationsdaten besitzen, kann es leicht zu einem nicht mehr einfach verwaltbarem Netz kommen. Dies ist durchaus realitätsnah. Ein Beispiel wäre das eines Haushaltes, der im Besitz einer IP-Kamera ist und zusätzlich noch einen Router besitzt. Bereits hier müssen zwei Geräte konfiguriert werden. Die Konfiguration geschieht in der Regel über integrierte Web-Server. Somit muss der Nutzer zwei Zugangsinformationen für zwei Geräte bereithalten. Bei einer Allgegenwart von intelligenten Gegenständen nach Maß des ***„Ubiquitous computing“*** wären dies dutzende an Authentifikationsdaten. Gewünscht wäre eine zentrale Konfigurationsstelle.

Beide Ansätze, die angesprochen wurden, besitzen unzufriedenstellende Merkmale. Einen PC als Gateway zu nutzen ist zu kostenintensiv und die Alternative der „Embedded Internet Computing“-Geräte besitzt ebenfalls einige nicht zu verachtende Schwachstellen. Gesucht wird nach einem Konzept, das Vorteile beider Ansätze nutzt und die Schwachstellen versucht zu minimieren. Die Ziele sind klar definiert:

- Hohe Verfügbarkeit
- Geringe Kosten
- Hohe Flexibilität und Erweiterbarkeit
- Generische Schnittstellen zum Internet

Hohe Verfügbarkeit

Die Allgegenwart der Informatik macht uns immer abhängiger von ihrer Funktionsfähigkeit. Wenn Abläufe im Alltag davon abhängen, ob Geräte einsatzbereit sind oder nicht, so führt dies unweigerlich zu hohen Ansprüchen an die Softwarequalität der Geräte. Besonders in vernetzten Systemen, drückt sich Softwarequalität direkt in Ausfallzeit aus.

Es wurde bereits auf zwei mögliche Konzeptionen für die Anbindung von Geräten an das Internet eingegangen. Die direkte Anbindung stellt ein wesentliches Sicherheitsrisiko dar. Die relativ beschränkten Ressourcen der eingesetzten Mikrocontroller, sind nicht ausreichend, um mehrere Benutzer adäquat zu bedienen. Weiterhin sind Angriffe auf solche Geräte verhältnismäßig einfach. Durch die beschränkte Rechenkapazität kann man den Mikrocontroller schnell durch ICMP-floods oder TCP-SYN-floods überlasten und damit funktionsuntüchtig machen. Ein weiterer möglicher Ansatz für DoS-Angriffe auf diese Geräte wird durch die Komplexität der Protokolle selbst eingeführt. Komplexe Netzprotokolle wie z.B. TCP/IP oder Ethernet machen besonders low-level-programmierte Mikrocontroller fehleranfällig. Solche Programmfehler könnten für DoS-Angriffe ausgenutzt werden. Dieser Sachverhalt verschlimmert sich dadurch, dass bei einer Vielzahl von Geräten unterschiedlicher Hersteller, eine Vielzahl von unterschiedlichen IP-Stack-Implementierungen existieren, jedes mit potentiellen Schwachstellen. Durch letztere Argumente ist die Überzeugung gereift, dass eine direkte Anbindung von Knoten an das Benutzernetz (z.B. Internet) vermieden werden sollte. Diese Erkenntnis soll für den Rest der Diplomarbeit vermerkt werden.

Konzeptionskriterium für das Rahmenwerk:

- Keine direkte Anbindung der Knoten an das Internet.

Das alternative, besprochene Konzept, das aktuell gerne eingesetzt wird, besitzt ebenfalls Schwachstellen: Das Benutzen eines PC Systems als Gateway zwischen Knoten und WAN ist ein guter Ansatz, um die Knoten zu schützen. Allerdings entsteht dadurch ein „Single point of failure“. Durch komplexe Betriebssysteme und komplexe Hardware, leiden PC basierende Systeme unter einer zu geringen Ausfallsicherheit. Optimalerweise wären 100% Verfügbarkeit das anzustrebende Ziel. Aus diesem Wunsch heraus entsteht ein weiteres Entwurfskriterium für das zu planende Rahmenwerk.

Konzeptionskriterium für das Rahmenwerk:

- PC basierendes System als Gateway vermeiden

Ein weiterer Punkt, der wichtig in Zusammenhang mit Ausfallsicherheit ist, ist der der Kommunikation. Welche Kommunikationskanäle zwischen Knoten und Internet sollten eingesetzt werden? Komplexe Netzwerktechnologien beeinflussen die Stabilität für einfache Zwecke negativ. Die bereits erläuterten Gründe hierfür werden noch mal kurz in Erinnerung gerufen. Hohe Ansprüche an die Software von Knoten könnte vermehrt zu Fehlern im Code führen. Der direkte Zusammenhang von Fehleranzahl und Programmgröße ist erwiesen. Mit dieser Erkenntnis ergibt sich ein weiteres Konzeptionskriterium.

Konzeptionskriterium für das Rahmenwerk:

- KISS, keep it short and simple

Das KISS-Prinzip vereinfacht das Auffinden von Fehlern im Rahmenwerk dadurch, dass die Zustandsmenge kleiner ist als die komplexerer Systeme und dadurch das Auftreten von Fehlern sich schneller manifestiert und diese somit schneller behoben werden können.
Besonders den Kommunikationsprotokollen sollte große Aufmerksamkeit gewidmet werden. Sichere Protokolle sind das Rückgrad für sichere Kommunikation. So sollten nur bewährte Protokolle und Verfahren zum Einsatz kommen, für die aktuell keine Schwachstellen bekannt sind. Dies führt zu einem weiteren Konzeptionskriterium.

Konzeptionskriterium für das Rahmenwerk:

- Bewährte Kommunikationsprotokolle, Kommunikationsverfahren und Kommunikationsalgorithmen verwenden.

Ein weiterer Aspekt ist die der Softwareprogrammierung. Bisweilen wurde nur über Ausfallsicherheit in Verbindung mit Technologien oder Konzepten diskutiert. Die Rolle die dem Softwareentwurf zukommt, ist nicht minder bedeutend. Prinzipiell sollte, sofern möglich, auf fehleranfällige Softwaretechniken verzichtet werden. Auch wenn zu diesem Zeitpunkt noch kein Entwurf existiert, sollen die wichtigsten Tugenden bereits festgehalten werden.

Konzeptionskriterium für das Rahmenwerk:

- Nebenläufigkeit, dynamische Allokation und unbewährte Komponenten vermeiden.

Nebenläufigkeit führt oft zu unerwarteten Folgen. Dies liegt nicht an der Nebenläufigkeit selbst, sondern eher an Entwicklungsfehlern durch den Programmierer. Ein System, das nach allen Regeln der Kunst entworfen wurde, wird auch dann korrekt funktionieren, wenn es massiv auf Nebenläufigkeit basiert.

Dynamische Allokation erhöht die Menge an Zuständen, in die ein Programm gerät. Laufzeitprobleme sollten von vornherein feststehen und bereits im Entwurf mit berücksichtigt werden. Ein plötzliches „out of memory“ verkompliziert die Fehlerhandhabung wesentlich. Ein frühes Festlegen auf Speicherzuteilungen ist hier ratsam.

Es sollen bei der Implementierung der Software nur solche Komponenten eingesetzt werden, die sich bewährt haben. Das Nutzen von Komponenten deren Reifegrad nicht bekannt ist, sollte nicht in Betracht gezogen werden.

Diese drei Kriterien gelten nur als Richtlinien, das strikte Einhalten wird nicht um jeden Preis erwünscht.

Geringe Kosten

Aus der Definition von *„Ubiquitous computing"* lässt sich folgern, dass der Mensch in Zukunft öfter mit intelligenten Geräten in Berührung kommt, als es aktuell der Fall ist. Die große Anzahl der Geräte, die mit Mikrochips ausgestattet werden, wird noch wesentlich zunehmen. Diese Entwicklung bedeutet weitere Kosten für den Besitzer dieser Geräte. Es sollte durch geeignet gewählte Konzepte versucht werden, die Kosten vor allem im Anschaffungspreis zu senken.

Auch in Hinsicht auf die Kosten ist das Konzept, in dem jedes Gerät über große Fähigkeiten verfügt z.B. Web-Server, unvorteilhaft. Komplexe Software wie Webserver und IP-Stack erfordern wesentlich mehr Ressourcen, z.B. RAM, Rechengeschwindigkeit und vieles mehr. Dies schränkt zum einen die Auswahlmenge geeigneter Mikrochips ein und zum anderen heben sich zusätzlich noch die Kosten pro Chip. Verzichtet man in weiten Teilen auf komplexe Fähigkeiten und überlässt einer anderen Instanz diese Aufgaben, so lassen sich die Kosten erheblich senken. Im einzelnen mögen die Kostenvorteile nicht so hoch sein, allerdings wird dieser Sachverhalt deutlicher bei einer großen Anzahl an Geräten, so wie es in der Vision von Marc Weiser prophezeit wurde. Aus den letzteren Überlegungen ergibt sich ein weiteres Konzeptionskriterium.

Konzeptionskriterium für das Rahmenwerk:

- Auslagerung der "Intelligenz" aus den Knoten

Um eine weitere Kostensenkung zu erreichen, sollte die Wahl der Technologien auf solche fallen, die weiter Verbreitung unterliegen. Technologien, die weit verbreitet sind, sind in der Regel durch Massenproduktion kostengünstiger. Weiterhin ist dies auch ein Argument das für die Ausfallsicherheit gilt, verbreitete Technologie ist in der Regel ausgereifter.

Konzeptionskriterium für das Rahmenwerk:

- Technologien nutzen, die weiter Verbreitung unterliegen

Ein weiterer Gedanke gilt der Kommunikation. Je einfacher die Kommunikation der Knoten, um so weniger Anforderungen werden an das Kommunikationsmedium gestellt. Relativ komplexe Technologien wie ATM oder Ethernet stellen größere Anforderungen an das Können der Knoten. Da Knoten keinen großen Durchsatz besitzen und auch die Antwortzeiten nicht kritisch sind, wäre die Wahl einer sehr einfachen Technologie vorteilhaft. Auch hier tritt wieder das KISS-Prinzip zu Tage, je weniger Ansprüche wir an die Knoten stellen, desto billiger werden auch die Einzelpreise.

Konzeptionskriterium für das Rahmenwerk:

- Einfache Kommunikationstechnologien wählen

Flexibilität und Erweiterbarkeit

Da das SASI-Rahmenwerk in erster Linie eine Middleware ist, bietet SASI nur Schnittstellen an die zur Kommunikation mit den Knoten genutzt werden können. In diesem Fall ist es notwendig, dass das Rahmenwerk so flexibel und erweiterbar ist wie möglich. Dadurch, dass Anwendungen erst durch den Nutzer entwickelt werden, darf das SASI-Rahmenwerk nicht einschränkend auf die Möglichkeiten wirken damit der Entwickler alle denkbaren Anwendungen realisieren kann.

Die Flexibilität betrifft in erster Linie die unterstützten Knoten und andererseits die Schnittstelle die der Klient-Applikation angeboten wird. Beide Aspekte werden im Kapitel der Architektur näher diskutiert. Die Erweiterbarkeit betrifft die Unterstützung von neuen Knotentypen. Das Rahmenwerk soll ohne hohen Entwicklungsaufwand um neue Knotentypen erweiterbar sein.

Schnittstelle zum Internet

Die Wahl der Service-Architektur und der Schnittstelle hin zur Nutzung des zu entwerfenden Rahmenwerks bildet ein zentraler Punkt der Problemstellung. Da das zu entwerfende System ein Rahmenwerk darstellt und nicht eine konkrete Applikation erfüllen muss, sollte die Schnittstelle möglichst offen und generisch sein. Mit offenen und generischen Schnittstellen, lässt sich später eine konkrete Applikation mit weniger Aufwand realisieren.

Die Wahl der Service-Architektur sollte ebenfalls eine möglichst einfache und zustandslose sein. Die Zustandslosigkeit vereinfacht Ad-Hoc Anfragen, die in einem ***„Ubiquitous computing"*** Szenario durchaus dem Normalfall entsprechen.

Konzeptionskriterium für das Rahmenwerk:

- Generische einfache und zustandslose Schnittstelle für Servicenutzung

Auch in diesem Fall empfiehlt sich, Technologien oder Konzepte zu nutzen, die weiter Verbreitung unterliegen. Dies vereinfacht das spätere Implementieren einer Applikation, die Dienste des Rahmenwerks nutzen soll. Werden Standards genutzt, die bereits in den meisten Sprachen als Module implementiert sind, führt dies zu einem vielleicht höherem Nutzungs- oder Erfolgsgrad des Rahmenwerks.

Zusammenfassung

Nachfolgend wird noch mal die Problemstellung und die daraus resultierenden Kriterien zusammengefasst.

Besprochen wurden zwei verbreitete Konzepte. Zum einen die Anbindung von Knoten an ein Gateway auf PC-Basis, zum anderen die direkte Verbindung aller Knoten an das Internet. Für beide Konzepte sprechen einige Vor– und Nachteile. Die Analyse dieser beiden Konzepte hat zu einigen Erkenntnissen geführt, die in Form von Konzeptionskriterien festgehalten wurden. Im Folgenden werden alle Kriterien noch einmal aufgelistet:

- Keine direkte Anbindung der Knoten an das Internet
- PC basierendes System als Gateway vermeiden
- KISS, keep it short and simple
- Bewährte und schwachstellenlose Kommunikationsprotokolle verwenden
- Nebenläufigkeit und dynamische Allokation vermeiden
- Auslagerung der Intelligenz aus den Knoten
- Technologien nutzen, die weiter Verbreitung unterliegen
- Einfache Kommunikationstechnologien wählen
- Generische, einfache und zustandslose Schnittstelle für Servicenutzung

Diese Kriterien werden während der Architektur- und Entwurfsphase als Richtlinien dienen.

Architektur

Basierend auf den ausgewählten Konzeptionskriterien soll das Rahmenwerk in diesem Teil des Dokumentes dargestellt werden. Trotzdem bleiben noch einige Entscheidungsfragen offen. Es wurde noch nicht geklärt, was genau dieses Rahmenwerk zu leisten im Stande sein soll.

Da es sich um ein Rahmenwerk handelt und beliebig ausgebaut werden kann, soll sich in dieser Diplomarbeit hauptsächlich auf die Funktionsfähigkeit der entworfenen Konzepte konzentriert werden. Konkrete Leistungsfähigkeit in einem bestimmten Applikationsfeld ist nicht das übergeordnete Ziel. Dennoch sollte zwecks Demonstration ein konkretes einfaches Anwendungsbeispiel realisiert werden.

Knoten

Die erste Frage, die sich stellt, ist die der Knoten. Was ist ein solcher Knoten und was genau ist er im Stande zu leisten? In diesem Rahmen sind solche Knoten die unterste Stufe ansteuerbarer Bauelemente. Diese Bauelemente sollen die Schnittstelle zwischen der Elektronik der Anwendung und der realen Welt bilden.

Da sich Gegenstände des Alltags anbinden lassen sollten, ist eine generische Schnittstelle wünschenswert. Aber was ist eine solche generische Schnittstelle? Jegliche physikalische Größe lässt sich mittels entsprechenden elektronischen Schaltungen in elektrische

Spannungen umwandeln. Mittels eines einfachen AD-Wandlers lässt sich nun die zu messende Größe digital erfassen. Da jegliche zu messende Größe durch eine entsprechende elektronische Schaltung erfasst werden kann, kann man davon ausgehen, dass ein AD-Wandler eine ausreichende Eingabemethode ist.

Jeglicher Knoten in diesem Rahmenwerk, der irgendeine Funktion erfüllt, erhält Eingaben in irgendeiner Form und drückt die Zustandsänderung durch Verändern von Ausgabegrößen aus. Ein Ausgabekanal in Form eines Schalters ist somit auch eines der wesentlichen Fähigkeiten eines Knotens. Neben diesen beiden einfachen Ein- und Ausgabemethoden soll noch ein Identifikationsmodul mit eingeplant werden. Es sei angemerkt, dass die Einfachheit dieser Knoten das Konzeptionskriterium der Kostenminimierung befriedigen. Einfache Schalter oder AD-Wandler-Bausteine sind in großen Stückzahlen zu Centpreisen erhältlich.

Nachdem nun definiert wurde, was Knoten für Funktionen erfüllen müssen, muss die Anbindung dieser Geräte an das Kommunikationsnetz bedacht werden.

Verwaltung

Laut einem der Kriterien sollte es vermieden werden, die Knoten direkt an ein IP-Netzwerk anzubinden, insbesondere dem Internet. Somit werden potentielle DoS-Angriffe von außen schon von vornherein vermieden. Das Nutzen eines Gateway ist somit unumgänglich. Laut einem anderen Kriterium sollten PC basierende Systeme als Gateway aus genannten Gründen vermieden werden. Dies führt direkt zur Wahl von kleinen Signalprozessoren oder Mikrocontrollern. Diese würden die Rolle übernehmen, die Knoten zu überwachen, abzufragen und zu konfigurieren, siehe Abbildung 3. Im folgenden wird dieser Mikrocontroller oder Signalprozessor mit Geräte-Controller bezeichnet. Bevor darauf eingegangen wird, wie dieser Geräte-Controller mit dem Internet verbunden wird, soll die Kommunikation zwischen Knoten und Geräte-Controller weiter vertieft werden.

Vernetzung

Da es sich um einfache Knoten handelt, ist die nötige Bandbreite des Kommunikationsmediums relativ gering. Komplizierte Adressierungsmechanismen sind ebenso überflüssig wie komplexe Flusskontrolle. Angesichts dieser Argumente und

Anforderungen scheint ein Bussystem geradezu optimal für diesen Zweck. Ein einfaches Bussystem genügt auch unserem Konzeptionskriterium des KISS-Prinzips. Ein Bussystem stellt in der Regel eine einfachere Technologie dar, als es z.B. eine Computernetzwerktechnologie tut. Der Entscheid, zu einfachen Knoten und einem ebenso einfachen Bussystem, genügt unserem Konzeptionskriterium der Verlagerung von Intelligenz weg von den Knoten hin zum Geräte-Controller, siehe Abbildung 3. Der Verzicht auf komplexe Netzwerktechnologie verhindert auch den Einsatz von teureren Treiberbausteinen für die Knoten.

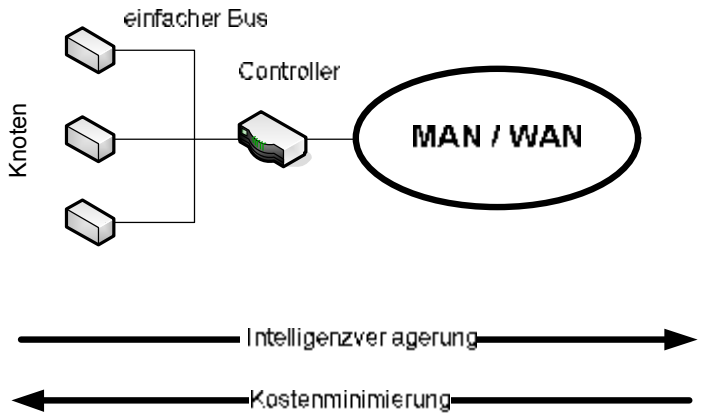


Abbildung 3 Knoten und Geräte-Controller

Die Peer-Applikation

Im Folgenden soll auf eine weitere Strukturkomponente eingegangen werden. Aus bereits erwähnten Zuverlässigkeitsgründen sollte auf direkte Anbindung von Mikrocontrollern an das Internet verzichtet werden. Die begrenzten Ressourcen können DoS-Angriffe begünstigen. Es bedarf somit einem Element, das den Mikrocontroller und das Internet miteinander verbindet. Da dieses Element nicht nur die Brücke zwischen Mikrocontroller und Internet schlagen wird, sondern auch andere Funktionen erfüllen wird, wird dieses Element im Folgenden, Peer-Applikation genannt. Der Geräte-Controller stellt somit nur eine Vorstufe dar, er regelt lediglich den Bus und die Knoten. Alle Informationen des Geräte-Controllers werden dann an die Peer-Applikation geschickt, wo sie dann dem Nutzer in aufbereiteter Form zur Verfügung gestellt werden.

Eines der Konzeptionskriterien rät zur Vermeidung von PC basierenden Systemen aus Gründen der Ausfallssicherheit. Dieses Problem kann jedoch vermieden werden, indem Redundanz eingeführt wird. Konkret bedeutet dies, dass der Geräte-Controller zulässt, dass mehrere Peers gleichzeitig angemeldet sein können. Die Informationen des Geräte-Controller werden dann an alle Peers gleichzeitig übermittelt. Somit sind mehrere Instanzen von Peers mit allen Informationen versorgt. Alle Informationen sind somit von mehreren unabhängigen Rechnern aus verfügbar. Die Einführung einer Peer-Instanz hat auch den Vorteil, dass ein Grossteil der komplizierten Aufgaben vom Geräte-Controller in den Peer verlagert werden können. Diese Verlagerung von Code aus dem Geräte-Controller begünstigt die Stabilität des Geräte-Controller selbst, denn weniger Codevolumen bedeutet weniger Fehler. Da der Geräte-Controller ein eingebettetes Gerät ist, mit Laufzeiten von Wochen bis hin zu Monaten oder gar Jahren, wirken sich Fehler fatal aus. Um jedoch zu vermeiden, dass ein PC-System Vorrausetzung für den Betrieb des ganzen Systems ist, soll der Geräte-Controller auch komplett ohne Peers operativ bleiben und die Knoten weiter verwalten können. Die Peer-Applikation kann bei Bedarf gestartet und auch wieder beendet werden. Somit bedarf es keinem dedizierten PC-System, um die Funktionsfähigkeit der Knoten zu erhalten.

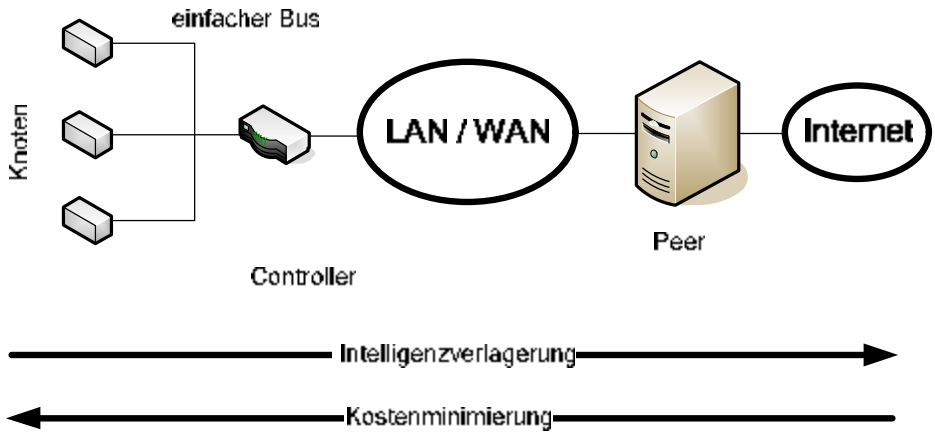


Abbildung 4 Peer und Geräte-Controller

Die Peer-Applikation bildet somit die Brücke zwischen Geräte-Controller und Internet, siehe Abbildung 4. Im Folgenden wird auf die Service-Schnittstelle des Peers eingegangen. Diese Schnittstelle definiert, wie auf Informationen der Knoten zugegriffen werden kann. Eines der Konzeptionskriterien legt fest, dass möglichst etablierte Technologien und Methoden verwendet werden sollen.

Vernetzung von Peer und Geräte-Controller

Bei der Auswahl der Technologie zur Vernetzung von Peer und Geräte-Controller, werden die Konzeptionskriterien berücksichtigt. Diese besagen, dass nur solche Technologien zum Einsatz kommen sollen, die weiter Verbreitung unterliegen und dadurch ausgereifter und günstiger auf dem Markt erhältlich sind. Die Wahl der Technologie wird auf den nächsten Teil der Diplomarbeit verlagert, in dem die Auswahl der Technologie diskutiert wird.

Klient-Applikation

Die Klient-Applikation ist die Applikation die vom Nutzer eingesetzt wird um eine bestimmte Anwendung zu realisieren. Die Klient-Applikation wird vom Nutzer entwickelt und benutzt die SASI-API zu Realisierung der angestrebten Anwendung. Die zur Verfügung gestellte SASI-API kapselt die Kommunikation zwischen Peer-Applikation und Klient-Applikation in einem Modul und ist aktuell für Python verfügbar. Die Kommunikation zwischen Peer-Applikation und Klienten ist maßgeblich durch die gewählte Service-Architektur geprägt die im folgenden Teilkapitel diskutiert wird.

Service-Architektur

Im Folgenden wird die Service-Architektur der Peer-Applikation, die für die Kommunikation zwischen Peer-Applikation und Klienten maßgeblich ist besprochen. Eine einfache Service-Architektur stellt REST dar. REST ist ein zustandsloses Client/Server Protokoll in der keine dauerhafte logische Verbindung besteht. Eine REST-Anfrage enthält alle notwendigen Information, um die Anfrage zu verstehen. Eine Anfrage besteht aus einer URI, die eindeutig einer Ressource zugeordnet werden kann. Zusätzlich zu dieser URI existiert noch der TYP der Aktion, die auf die URI angewendet werden soll. Optional können zu der angegebenen Aktion noch Daten mitgesendet werden. Solche Aktionen sind:

- POST: Diese Aktion überschreibt eine existierende Ressource mit den mitgeschickten Daten.
- PUT: Diese Aktion erstellt eine noch nicht vorhandene Ressource
- DELETE: Löscht eine existierende Ressource
- GET: Verlangt den Inhalt einer bestehenden Ressource

Die genannten Aktionen sind keinesfalls zwingend. Das REST-Prinzip kann nach Belieben angepasst werden. Zu beachten gilt, dass die REST-Aktionen in diesem Rahmenwerk von beiden Seiten erzeugt werden können. So kann z.B. der Klient der Peer-Applikation mitteilen, dass ein Schalter seinen Zustand ändern soll, dies würde in diesem Fall mittels einer POST-Aktion auf die URI des Schalters geschehen. Gleichermaßen kann die Peer-Applikation auch eine REST-Aktion an den Klienten richten. Ein Beispiel hierfür, wäre ein eingetretenes Ereignis, wie die Betätigung eines Schalters. Da eine solche REST-Aktion nicht dem Sinn von POST, PUT, DELETE oder GET entspricht, ist es sinnvoll die NOTIFY-Aktion einzuführen. Diese würde eine Änderung, bezogen auf eine URI mitteilen.

Das REST-Prinzip ist nicht neu, so wird REST bereits erfolgreich in HTTP und SNMP eingesetzt. Diese weite Verbreitung und die Einfachheit von REST befriedigt zwei Konzeptionskriterien zugleich, nämlich: KISS und das Verwenden von verbreiteten Technologien. Durch die Zustandslosigkeit der Kommunikation unterstützt dieses Prinzip auch Ad-Hoc-Anfragen wie sie im Verwendungsfeld dieses Rahmenwerks durchaus den Normalfall darstellen. So kann z.B. ein Klient eine POST-Aktion an die Peer-Applikation richten, ohne vorher am Peer angemeldet zu sein.

Gesamtarchitektur

Motiviert durch die Konzeptionskriterien wurden die folgenden Architekturentscheidungen ermittelt.

- Verwenden von sehr einfachen Knoten (Schalter, AD-Wandler,...)
- Verwenden eines einfachen Busses zwischen Knoten und Geräte-Controller
- Der Geräte-Controller wird auf Basis eines preisgünstigen Mikrocontrollers realisiert
- Die Peer-Applikation basiert auf PC-Systemen
- Redundanz auf Peer-Ebene sollte mit eingeplant werden
- Kommunikation zwischen Klient und Peer sollte auf dem REST-Prinzip basieren
- Einfache Netztechnologie zwischen Geräte-Controller und Peer

In Abbildung 5 sind alle Architekturentscheidungen zu einer gesamten Struktur zusammengefasst.

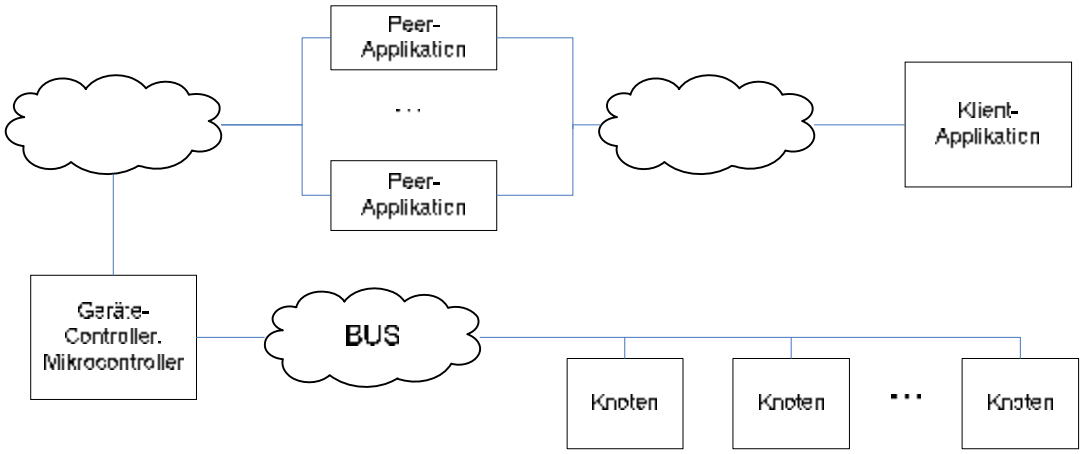


Abbildung 5 Architektur

Andere Lösungskonzepte

An dieser Stelle wird auf Projekte eingegangen, die ähnliche Ziele verfolgen wie die in dieser Diplomarbeit. Als erstes wird auf grundlegende Konzepte eingegangen, die bereits etabliert sind und durchaus auch in der Industrie Verwendung finden.

SCADA

Ein in der Industrie geläufiges Konzept ist das so genannte SCADA [ICITA1]. SCADA steht für "Supervisory Control and Data Acquisition". Es handelt sich hierbei um ein Konzept zur Erfassung, Steuerung und Überwachung von Sensoren und Aktuatoren. Es wird hauptsächlich zur Steuerung von technischen Prozessen angewandt. Es wird in diesem Kontext zur Sprache gebracht, da es in dieser Diplomarbeit im weitesten Sinne auch um Erfassung von Daten und Steuerung von Elementen geht.

Bei SCADA werden Sensoren und Aktuatoren von Remote-Terminal-Units oder von Programmable-Logic-Controller verwaltet. Diese Einheiten führen auch eine Aggregation von Daten aus. Die Steuerung und Datensammlung wird von so genannten Master-Stationen übernommen. Bei einer solchen Master-Station, handelt es sich um einen PC, der auch eine Benutzerschnittstelle anbietet.

Bei SCADA handelt es sich vor allem um ein Konzept und nicht um eine fertige Technologie. Es gibt diverse Anbieter, die SCADA-Lösungen anbieten, allerdings sind diese oftmals proprietär. Somit sind die konkret eingesetzten Mittel oft variabel. Beispielsweise ist die Kommunikation zwischen den einzelnen Sensoren und Aktuatoren nicht fest an eine Technologie gebunden, wobei aber Ethernet und IP oft in Erscheinung treten.

Ein Schlagwort bei diesem Konzept ist auch das so genannte Web-Scada. Hierbei wird das Human-Machine-Interface über das WWW zur Verfügung gestellt. Auch hier gibt es diverse Anbieter einer solchen Technologie.

Prinzipiell soll SCADA andere Bedürfnisse als die in der Problemstellung genannten befriedigen. Dies erklärt auch den vermehrten Einsatz von Remote-Terminal-Units, mit denen höhere Kosten als die hier tolerierten einhergehen. Dennoch kristallisiert sich aus SCADA der Ansatz heraus, dass Knoten von Geräte-Controllern überwacht und geregelt werden. Diese Aufgabe wird in SASI vom Geräte-Controller durchgeführt, zwar nicht in Form eines Programmable-Logic-Controller-Mikrochips sondern in Form eines Mikrocontrollers. Dies ist nicht die einzige Analogie zwischen SASI und SCADA. So ist der Trend hin zu Web-SCADA auch ein fester Bestandteil des SASI-Rahmenwerks.

WSN

Ungleich dem Vorgehen in dieser Diplomarbeit trifft man häufig bei Sensor-Netzwerken auf das so genannte WSN. WSN steht hier für "Wireless Sensor Network". Hierbei will man von vornherein Sensoren und Aktuatoren drahtlos anbinden. Jeder Knoten in einem solchen WSN besitzt eine Radioeinheit, um mit dem drahtlosen Netzwerk zu kommunizieren. Die Kommunikation zwischen den einzelnen Knoten ist nicht auf ein Standard festgelegt. Allerdings wird gerne auf den ZigBee-Standard zurückgegriffen.
Solche Produkte werden bereits vertrieben. Ein solcher Hersteller ist z.B. Crossbow [2]. Dessen "Wireless Measurement System Mica 2", ist ein solcher Knoten, der eine generische Schnittstelle zum Sensor / Aktuator besitzt, z.B. UART, I2C und andere. Die Herstellung des

[2] http://www.xbow.com/

Aktuators / Sensors liegt somit in der Hand des Entwicklers. Weiterhin wird von Crossbow die Basisstation vertrieben, die es erlaubt, alle Daten des WSN zu sammeln und an einen Rechner weiterzuleiten. Schnittstellen der Basisstation sind z.B. USB oder Ethernet mit TCP/IP. Mit GSN[3], einem Opensource-Middleware-Projekt des LSIR Lab für Sensornetzwerke kann man dieses WSN inspizieren, manipulieren und auswerten.

Interessant sind auch Versuche, jeden dieser Knoten IP-fähig zu machen [IAB1]. Es wird versucht, die Knoten nicht über einen Proxy oder Gateway an das IP-Netzwerk anzuschliessen, sondern gleich jeden Knoten IP-fähig zu machen. Allerdings setzen diese Ansätze IPv6 voraus, weiterhin sind Sicherheitsaspekte hier offene Fragen, wie z.B. Schutz vor DoS-Angriffen oder die Regelung der Zugangskontrolle.

Prinzipiell ist das Konzept jeden Knoten IP-fähig zu machen, dem des SASI-Rahmenwerks gegenläufig. Eine der Architekturentscheidungen dieser Diplomarbeit bedingt die Intelligenzverlagerung, weg von den Knoten hin zu einem Verwalter der in dieser Diplomarbeit Geräte-Controller genannt wird. Bei diesem Projekt werden ungleich mehr Fähigkeiten von den Knoten gefordert. Dies führt zu erheblichen Mehrkosten im Vergleich zu SASI. Die Frage, ob jeder Knoten nun drahtlos angebunden werden muss hängt natürlich vom Applikationsfeld ab.

INSIGHT

INSIGHT steht für "Internet-Sensor Integration for Habitat Monitoring" [INSIGHT]. Es ist ein konkretes Projekt, in dem Sensoren drahtlos im häuslichen Bereich Informationen nach außen zur Verfügung stellen. Es gibt einige Analogien zu SASI. In diesem Projekt wurden die bereits erwähnten Komponenten von CrossBow verwendet. Diese Knoten übermitteln ihre Daten an die Basisstation, wo eine Aufbereitung der Daten stattfindet. Die Daten werden dauerhaft gespeichert und können über eine Web-XML-Schnittstelle abgefragt werden.

[3] http://gsn.sourceforge.net

Die in dieser Diplomarbeit entwickelte Plattform besitzt ebenfalls diese Form von Schnittstelle. Interessant an INSIGHT sind die Kosten die hierbei bei ungefähr 100$ pro Knoten liegen, wogegen bei SASI diese sich im Bereich von 1$ bewegen.

Auch hier sind die Kosten pro Knoten sehr hoch. Die Ausstattung einer Wohnung nach Maß des *„Ubiquitous computing"* impliziert erhebliche Kosten. Weiterhin bedeutet eine drahtlose Verbindung auch, dass die Knoten über eine eigene Energiequelle verfügen müssen. Dies führt dazu, dass in jedem Knoten nach einer gewissen Zeit, die Bewohner oder ein Dienstleister alle Energiequellen austauschen muss.

INTERNET0

Bei Internet0 [MIT1] handelt es sich um eine interessante Initiative, in der man jegliches Objekt in ein IP-fähiges Endgerät verwandeln will. Anders hier, ist der grundlegende Gedanke, alle "bürokratischen" Hürden zu überwinden und weniger Code zu erzeugen. Wenn man Schalter oder Lampen IP-fähig machen will muss man den Entwurfsbürokratismus überwinden, so wird bei I0 z.B. auf Ethernet als Zwischenschicht komplett verzichtet. Weniger ist in diesem Kontext mehr, da es die Vorrausetzungen an die Chips in den Geräten verkleinert. Bei I0 sollen Netzwerkknoten eigenständig und nicht auf einen Master angewiesen sein. Ein Beispiel eines Schalters der in I0-Manier entwickelt wurde, besitzt lediglich 3.5kb Codegrösse inklusive TCP/IP-Stack und Web-Server.

Dieses Konzept besitzt eine große Schnittmenge mit dem hier entwickelten SASI-Rahmenwerk. Zwar geht es bei Internet0 nicht um Intelligenzverlagerung, sondern um Intelligenzminimierung, was in diesem Fall zu den gleichen Zielen führt. Jedoch sollte berücksichtigt werden, dass eine direkte Anbindung von Geräten an das Internet nicht von Vorteil aus Sicht der Sicherheit ist. Dies wurde bereits mehrmals in den vorangegangenen Kapiteln gezeigt.

GSN

GSN steht für "Global Sensor Network" und wurde an der EPFL[4] in Lausanne, Schweiz, entwickelt. Bei GSN handelt es sich um eine generische Middleware-Software, um Sensoren IP-fähig zu machen. Es gibt besondere Wrapper, die Sensordaten in das GSN einspeisen. So können die Daten z.B. über ein Serial-Interface, UDP und viele mehr zu GSN gelangen. Sensoren werden in Form von XML-Dateien vorkonfiguriert. Dabei handelt es sich um so genannte virtuelle Sensoren. Die technische Verbindung mit den physikalischen Sensoren wird teilweise mit Hilfe des IEEE-1451-Standards, automatisch abgewickelt. Dies unterstützt das "Zero-Programming"-Konzept. Die Sensordaten werden aufbereitet und über einen Web-Server zur Verfügung gestellt. GSN stellt auch andere Kommunikationsmittel wie Email oder SMS zur Verfügung.

Besonders herausragend an diesem Projekt ist die absolut unabhängige Geräteschnittstelle. Es Bedarf keinem Programmieraufwand, um neue unbekannte Sensoren in dieses Rahmenwerk einzubinden. Dies hat den Vorteil, dass der Anwender keine Änderungen an der Middleware vornehmen muss, um neue Sensoren zu unterstützen. Allerdings können Sensoren, die diesen Standart nicht erfüllen, nicht eingesetzt werden.

SASI

Die Herangehensweise in dieser Diplomarbeit besteht darin, in dem zu entwickelnden Rahmenwerk SASI ein einfaches Bussystem zu verwenden, um Knoten an eine Master-Station anzubinden. Dieser Bus wird an einen Master angeschlossen, der IP-Fähigkeit besitzt. Tendenziell lässt sich feststellen, dass in den genannten verwandten Projekten versucht wird, den einzelnen Knoten selbst mehr Fähigkeiten zuzuführen. Dies kann von Datenaggregation bis hin zu Wireless-Fähigkeiten reichen. Es lässt sich auch feststellen, dass so genannte Feldbusse durchaus gerne in Ethernet-Netze gebrückt werden. Insgesamt verfestigt sich der Eindruck, dass auch für einfache Sensoren oder Aktuatoren um neueren Konzepten standhalten zu können, eine große Intelligenz vorausgesetzt wird, Stichwort "Smart Sensors" [SI1].

---

[4] http://lsirwww.epfl.ch/

Da das SASI-Rahmenwerk davon ausgeht, dass es einer sehr großen Zahl an Sensoren oder Aktuatoren gegenübersteht, war eines der Ziele die Kostensenkung. Die wenigsten der genannten Projekte unterstützten diese Annahme als sinnvolles Ziel. Ein weiteres Merkmal von SASI, sollte die Verfügbarkeit und Kommunikationssicherheit sein. Bei genannten Projekten erschien der Trend, Knoten drahtlos und gleichzeitig IP-fähig zu machen. Allerdings schien es nicht klar, wie hohe Sicherheitsansprüche in diesen Szenarien realisiert werden.

Analog zu dem GSN-Projekt, bietet auch SASI eine API an. Durch die Peer-Applikation bietet das SASI-Rahmenwerk den Klienten eine bequeme Dienst-Schnittstelle, mit der die Klient-Applikation auf die Knoten zugreifen kann. Somit lässt sich SASI in die Kategorie der Middleware-Projekte einordnen, da SASI ohne Klient-Applikation keine praktische Anwendbarkeit besitzt.

Technologiewahl

In diesem Kapitel wird auf die Auswahl von Technologien oder Produkten eingegangen. Bei der Auswahl der Technologien oder Produkte werden die im letzten Kapitel ermittelten Architekturentscheidungen berücksichtigt. Es sei angemerkt, dass die Konzepte von denen SASI Gebrauch macht technologieunabhängig sind. Folgende Auswahl der Technologie dient nur der Realisierung eines Demonstrators um die Funktionsfähigkeit von SASI nachzuweisen.

Knoten

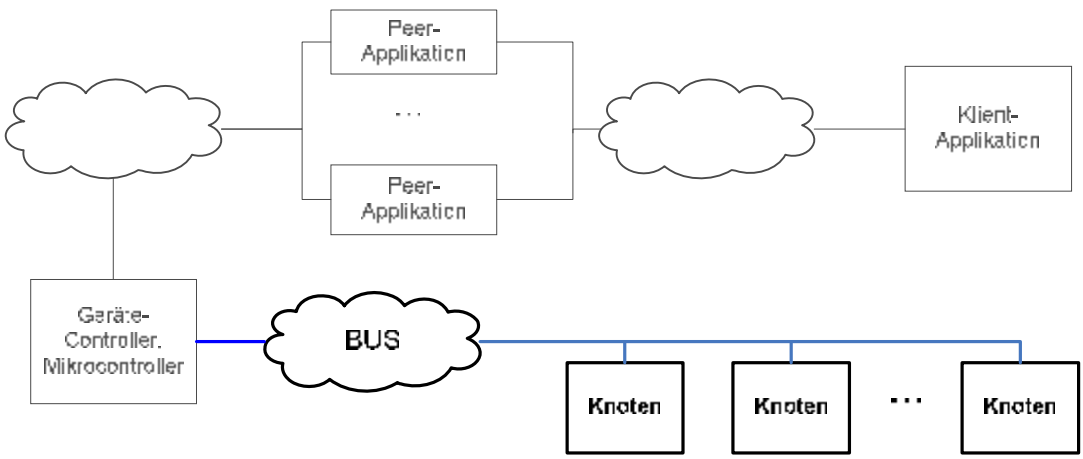


Abbildung 6 Knoten in der Gesamtstruktur

Die erwähnten Knoten wurden so definiert, dass diese einfachste elektronische Aufgaben erfüllen können. Ausgegangen davon, dass jede physikalische Größe in elektrische Spannungen umgewandelt werden kann, sollte die Wahl von AD-Wandlern als unbedingt

notwendig erachtet werden. Zwecks digitaler Ein- und Ausgabe von Daten wird ein einfaches elektronisches MOSFET genutzt, an dem zusätzlich die angelegte Spannung als High oder Low erfasst werden kann. Somit kann man analoge Spannungswerte einlesen, digitale Ausgabe mittels des MOSFET-Schalters realisieren und zusätzlich noch digitale Eingaben erfassen. Die zu implementierende Funktion des Gerätes wird schlussendlich auf Schaltkreisniveau realisiert.

In diesem Teil werden zur Verfügung stehende Produkte miteinander verglichen, um schlussendlich eine geeignete Wahl zu treffen. Laut den Architekturentscheidungen sollten diese Geräte busfähig sein und zugleich wenig Kosten verursachen. Im Folgenden soll die Wahl der Knoten in Funktion der Busauswahl getroffen werden. Bussysteme sind zum Vergleich besser geeignet als die Sensoren oder Aktuatoren selbst, weil für jeden Bus gibt es AD-Wandler oder E/A-Geräte. Die Geräte unterscheiden sich hauptsächlich durch die Eigenschaften die der Bus mit sich bringt. Angesichts der großen Anzahl von Bussen die auf dem Markt existieren, werden hier nur drei bekannte Vertreter vorgestellt.

CAN-Bus

CAN steht für „***Controller Area Network***“ und bildet einen Bus zur Kommunikation mit Sensoren und Aktuatoren. Ursprünglich wurde der CAN-Bus von Bosch eingeführt, um die Kabellänge in Automobilen zu reduzieren. Der CAN-Bus erfreut sich durch seine Ausgereiftheit großer Verbreitung. Dieser Bus benutzt 2 Leitungen für die Kommunikation und 2 für die Energieversorgung.

I2C-Bus

I2C steht für „***Inter-Integrated Circuit*“**. Es handelt sich hierbei um eine Entwicklung von Philips. Der I2C-Bus kommt zum Einsatz, um elektronische Komponenten miteinander zu verbinden. Allerdings sind hier kleine Übertragungsraten vorausgesetzt. Weiterhin kann der I2C-Bus nicht über Distanzen von mehreren Metern kommunizieren. Ursprünglich war der I2C-Bus dazu entwickelt worden, um lokal innerhalb eines Gerätes zu agieren. Zur Kommunikation benötigt der I2C-Bus lediglich zwei Drähte.

1W-Bus

Der 1W-Bus ist eine proprietäre Lösung von Maxim-IC[5]. Dieser Bus dient als Kommunikationsmedium für 1W-Geräte [MAX1], die ebenfalls von Maxim-IC hergestellt werden. Es gibt keinen bestimmten Verwendungszweck für diesen Bus. Genau hier liegt seine Stärke. Der Bus kann über eine maximale Distanz von 300 Metern noch mit seinen Geräten kommunizieren. Weiterhin benötigt dieser Bus lediglich 2 Leitungen. Die angeschlossenen 1W-Geräte können die benötigte Energie von der Kommunikationsleitung abzweigen. Die Übertragungsrate liegt bei nur 16,3 Kbit/s. Dies scheint recht langsam, aber angesichts des Verwendungszwecks ist dies immer noch ausreichend. Als weiteren Vorteil dient die hohe Anzahl an verfügbaren 1W-Geräten. Diese Geräte erfüllen diverse Applikationszwecke. Erhältlich sind z.B.: Schalter, AD-Wandler, Temperatursensoren, RAM-Module und viele andere.

Vergleich der Busse

Angesichts der genannten Vor- und Nachteile von CAN, I2C und dem 1W-Bus, fällt die Entscheidung eindeutig auf den 1W-Bus. Angesichts der nur zwei Leitungen und der parasitären Strombetreibung der Geräte, unterstützt der 1W-Bus zusätzlich noch große Distanzweiten. Die preiswerten 1W-Geräte die genau die gewünschten Zwecke erfüllen, sind dabei noch ein Grund mehr sich für dieses Produkt zu entscheiden.

Betrieben wird der 1W-Bus mit einem Master auf einer zweidrahtigen Leitung. Die so genannten Slaves, stellen die Knoten dar. Diese Knoten sind alle parallel geschaltet und werden von einem Master geregelt. Slave-Geräte dürfen den Bus nie von sich aus nutzen. Der Master ist in der Lage, den Bus zu scannen oder ein einzelnes Gerät anzusprechen und gegebenenfalls zu programmieren. Es ist somit wünschenswert, dass der Geräte-Controller, der zum Einsatz kommt, über einen solchen 1W-Master verfügt. Alternativ können 1W-Master-Bausteine über eine serielle Schnittstelle mit dem Geräte-Controller verbunden werden, siehe Abbildung 7.

---

[5] www.maxim-ic.com

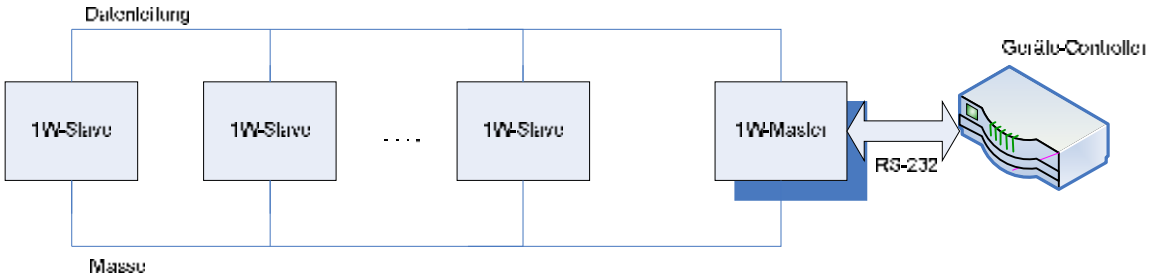


Abbildung 7 1W-Master und -Slaves

Geräte-Controller und Vernetzung

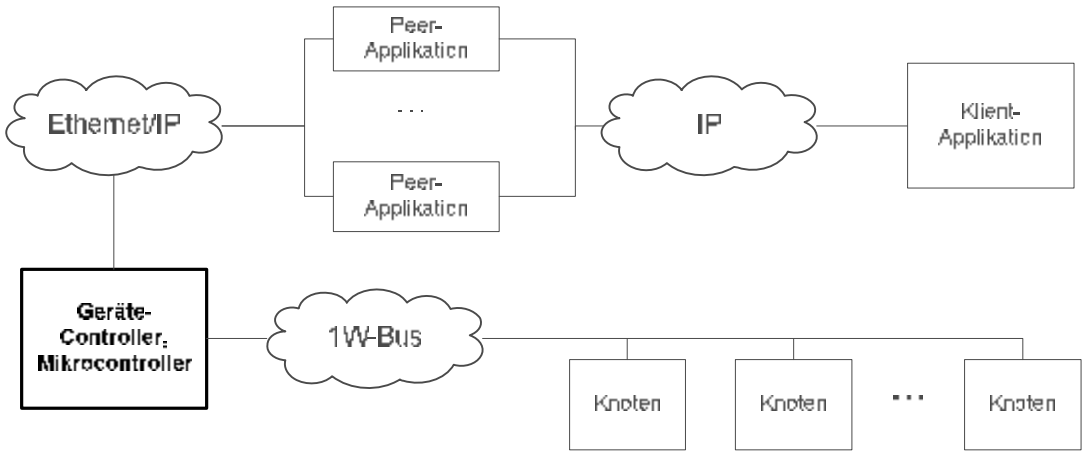


Abbildung 8 Geräte-Controller in der Gesamtstruktur

Nach der Auswahl der Knoten und des Busses, steht die Auswahl des Geräte-Controllers an erster Stelle, siehe Abbildung 8. Aus bereits erwähnten Gründen sollte es sich beim Geräte-Controller um ein einbettbares Gerät handeln. Idealerweise ein Mikrocontroller, DSP oder FPGA. Da die Aufgaben, die der Geräte-Controller übernehmen soll, zu komplex für FPGAs sind, wird auf diese verzichtet. Digitale-Signalprozessoren sind eher darauf ausgerichtet, kontinuierliche Ströme von Daten zu verarbeiten. Dies ist hier nur grenzwertig der Fall. Schlussendlich bleibt noch die Option eines Mikrocontrollers. Mikrocontroller erhält man in einer großen Variationsbreite. Somit ist die Chance gegeben, dass für den nötigen Verwendungszweck ein solcher Mikrocontroller existiert. Bevor jedoch auf die verfügbaren Mikrocontroller eingegangen werden soll, sollte die Kommunikationstechnologie zwischen Peer und Geräte-Controller besprochen werden, da dies ein beschränkender Faktor bei der Auswahl des Mikrocontrollers ist.

Laut Konzeptionskriterien soll eine direkte Anbindung des Geräte-Controller an das Internet vermieden werden. Stattdessen soll eine Art Gateway dazwischen geschaltet werden. Dieses Gateway wurde Peer genannt. Welche Kommunikationstechnologie zwischen Peer und Geräte-Controller zum Einsatz kommen soll, ist Thema des folgenden Abschnitts.

Die im Kapitel Problemstellung aufgestellte Liste an Konzeptionskriterien enthält ein Kriterium bezüglich Kommunikationstechnologien: Die Kommunikationstechnologie sollte weiter Verbreitung unterliegen und somit Ausgereiftheit und reduzierte Kosten implizieren. Die meist eingesetzte Kommunikationstechnologie bei Vernetzung von Rechnern ist Ethernet. Durch die weite Verbreitung sind alle Komponenten zum Aufbau einer Ethernet-Verbindung kostengünstig erhältlich. Dieser Sacherverhalt spiegelt sich auch in der Unterstützung von Ethernet in der Mikrocontroller-Welt nieder. Mikrocontroller mit der Option Ethernet Treiber anzuschließen sind weit verbreitet. Einige Mikrocontroller unterstützen Ethernet sogar direkt. Optimalerweise wäre bei der Wahl des Mikrocontrollers auf native Ethernet-Unterstützung zu achten. Die Verwendung von Ethernet als Kommunikationstechnologie ermöglicht es auch, mehrere Peers gleichzeitig ansprechen zu können. Zum Beispiel bei der Wahl von USB wäre dies nicht möglich gewesen. Im Folgenden werden einige Mikrocontroller mit ihren Vor- und Nachteilen vorgestellt.

RCM4000 RabbitCore

Der RCM4000 ist ein eingebettetes Netzwerk- und Kontrollmodul. Dieses Modul verfügt über 512K Flash und 512K SRAM. Über diese Eigenschaften hinaus, verfügt dieses Modul über diverse E/A-Schnittstellen. Entwickelt wird mit einem mitgelieferten C-Compiler. Preis dieses Moduls liegt bei 56$ bei einer Anzahl von 100 Stück.

MAXIM-IC

"Tini Networked" ist der Name eines Mikrocontroller vom Hersteller Maxim-IC. Dieser Mikrocontroller verfügt über einen integrierten 1W-Master sowie diverse andere E/A-Schnittstellen. Von Maxim-IC wird ein fertiges Modul angeboten. Dieses Modul verfügt ebenfalls über 512Kb RAM und 512Kb Flash. Der Preis liegt bei 110$ für ein komplettes Modul. Allerdings liegt der Preis dieses Mikrocontrollers bei nur 8$ bei einer Anzahl von 100 Stück.

Vergleich der Mikrocontroller

Neben den beiden genannten Mikrocontrollern existieren noch eine große Anzahl an anderen Produkten. Die meisten dieser nicht genannten Mikrocontroller verfügen jedoch nicht über eine Ethernet-Schnittstelle. Andere, mächtigere Mikrocontroller z.B. von Freescale besitzen solche Fähigkeiten, allerdings unter anderen Preisbedingungen.
Da der "Tini Networked" Mikrocontroller über einen dedizierten 1W-Master verfügt, fällt die Wahl auf das Produkt von Maxim-IC. Der "Tini Networked" Mikrocontroller ist kompatibel zum klassischen 8051-Mikrocontroller. Die Programmierung vom "Tini Networked" Mikrocontroller kann entweder in Java oder C realisiert werden. Von Maxim-IC selbst gibt es eine JVM, die auf dem "Tini Networked" lauffähig ist. Somit steht einem die ganze Mächtigkeit von Java zur Verfügung. Um eine maximale Leistung zu garantieren, wurde sich jedoch dazu entschieden, die Implementierung in C durchzuführen. Ein schneller IP-Stack kann Vorteile bei DoS-Angriffen bieten. Ein Teil des IP-Stacks wird bereits durch den Tini-ROM zur Verfügung gestellt.

Die Peer-Plattform

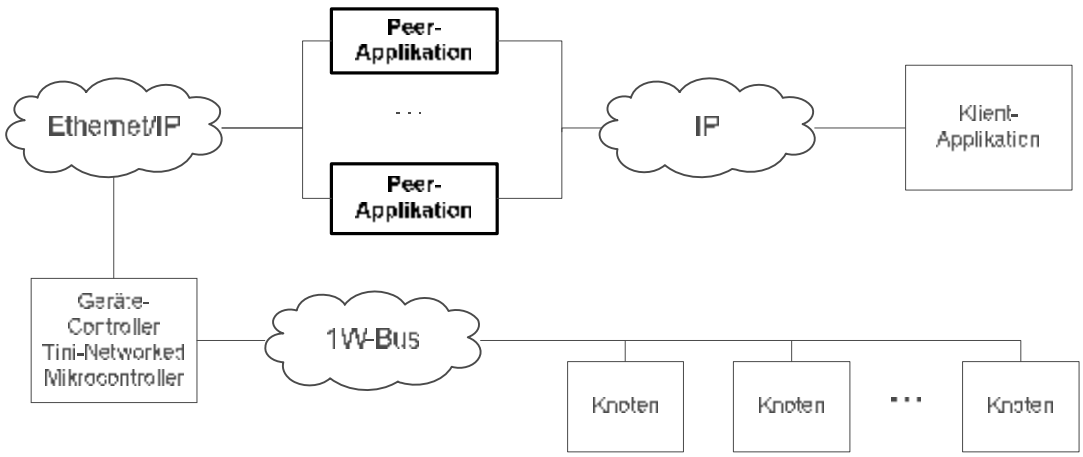


Abbildung 9 Peer-Applikation in der Gesamtstruktur

Schlussendlich fehlt noch die Entscheidung, auf welcher Plattform die Peer-Applikation implementiert wird, siehe Abbildung 9. Die Peer-Applikation sollte laut Konzeptionskriterium auf einem PC-basierten System lauffähig sein. Um maximale Portabilität zu erreichen, wäre eine Interpreter-basierte Sprache von Nutzen. In diesem Fall wurde sich für Python entschieden. Python bietet höchste Portabilität und eine gute Ausstattung bereits implementierter Module.

Zusammenfassung

Bevor zu dem Entwurf übergangen wird, sollten noch mal alle Konzeptionsideen und ausgewählte Produkte rekapituliert werden.

Es wurde sich in den vorherigen Kapiteln dazu entschieden, das Rahmenwerk in mehrere Teile aufzuspalten. Zum einen werden die Knoten über einen Bus an den Geräte-Controller angeschlossen. Dieser Geräte-Controller verwaltet den Bus und die Knoten. Bei Bedarf können sich über den Geräte-Controller so genannte Peers mit dem Geräte-Controller verbinden. Die Peer-Applikationen können über den Geräte-Controller Informationen über die Knoten beziehen oder die Konfiguration der Knoten ändern. Zwecks Redundanz sollte es erlaubt sein, mehrere Peers gleichzeitig mit dem Geräte-Controller zu verbinden.

Die oben erläuterte Struktur wird konkret mit folgenden Produkten und Technologien verwirklicht: Für die Knoten kommen "1W-Geräte" zum Einsatz. Gewünscht sind AD-Wandler, E/A-Geräte sowie ein Temperatursensor und ein ID-Chip. Diese vier Funktionalitäten werden von jeweils vier verschiedenen 1W-Geräten übernommen. Diese vier Knoten werden über den 1W-Bus mit dem Geräte-Controller verbunden. Der Geräte-Controller wird von einem Mikrocontroller-Modul von Maxim-IC realisiert. Auf diesem Modul kommt der "Tini Networked" Mikrocontroller zum Einsatz. Vorteilhafterweise verfügt dieser bereits über eine 1W-Bus-Master-Komponente, sowie über eine Ethernet-Schnittstelle. Aus genannten Gründen wird eine Ethernet-Verbindung zwischen Geräte-Controller und Peer-System bevorzugt. Das so genannte Peer-System wird aus Portabilitätsgründen in der Sprache Python realisiert. Um eine möglichst generische Schnittstelle zwischen Klient-Applikationen und dem Peer zu gewährleisten, wurde sich für das REST-Prinzip entschieden.

Im folgenden Kapitel wird genauer auf Entwurfsentscheidungen eingegangen. Zu diesem Zeitpunkt sind lediglich die Gesamtstruktur definiert und die Technologieauswahl abgeschlossen.

Entwurf der Kommunikation

In diesem Kapitel werden die Entwürfe des gesamten Systems genauer ausgearbeitet. Bislang wurden nur grobe Architektur- und Technologiefragen behandelt. Offen bleibt der Entwurf folgender Teilsysteme:

- Kommunikation zwischen Geräte-Controller und Knoten
- Kommunikationsprotokoll zwischen Geräte-Controller und Peer-Applikation
- Softwareentwurf für den Geräte-Controller
- Softwareentwurf für den Peer
- Datenstruktur zur Speicherung der Informationen im Peer
- Kommunikation von Klienten und Peer
- Entwurf eines Demonstrators

Zwar wurde REST als Prinzip für die Kommunikation zwischen Klienten und Peer bereits festgelegt, allerdings handelt es sich bei REST lediglich um ein Prinzip, es wird nötig dieses Prinzip in einer konkreten Schnittstelle bereitzustellen.

Da die Datenstruktur nicht direkt mit der Grobarchitektur zusammenhing, wurde bisher noch nicht auf die Datenstruktur für die Informationsspeicherung eingegangen. Diese Struktur wird in diesem Kapitel im Sinne von REST entworfen und definiert.

Schlussendlich wird ein einfacher Demonstrator entworfen, der die Funktionsfähigkeit des Rahmenwerks beweist.

Kommunikation zwischen Geräte-Controller und Knoten

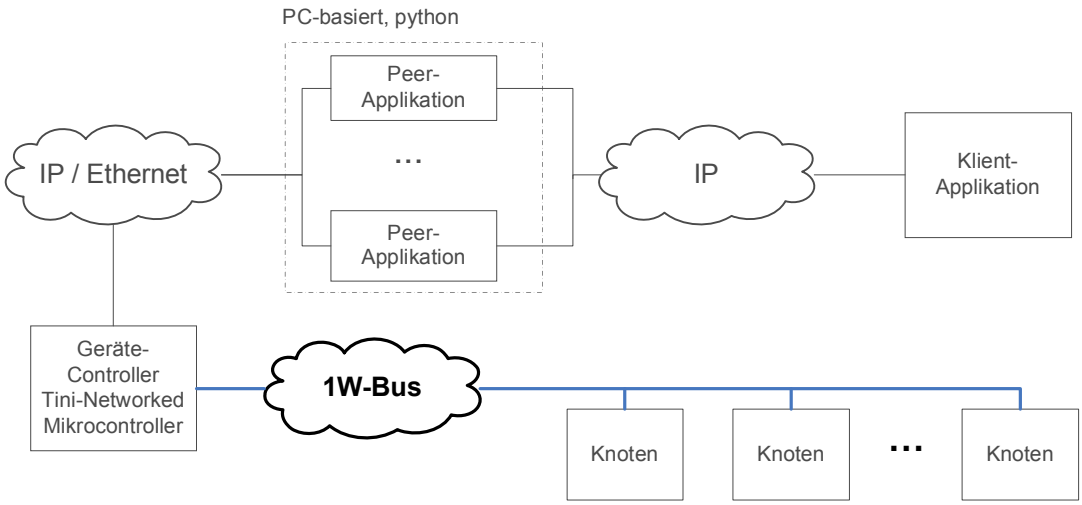


Abbildung 10 1W-Bus in der Gesamtstruktur

Für die Kommunikation zwischen den Knoten und dem Geräte-Controller wurde sich für den 1W-Bus entschieden, siehe Abbildung 10. Durch die Wahl dieses Busses wurde das Kommunikationsprotokoll schon implizit definiert. Es bleiben jedoch einige prinzipielle Fragen offen.

Der 1W-Bus bietet lediglich die Möglichkeit an: Bytes zu lesen und zu schreiben. Die Verwaltung der Geräte muss durch die Software realisiert werden. So muss z.B. das Suchen nach neu angeschlossenen Geräten explizit durch die Software angeordnet werden. Ebenfalls nötig wird das Ermitteln von Geräten, die den Bus verlassen haben. Da sich nach einigen Testläufen herausgestellt hat, dass die Suche nach neuen Geräten bis zu 300ms dauern kann, muss eine Lösung diesbezüglich gefunden werden. Ein Grund, wieso lang andauernde Zugriffe auf den Bus vermieden werden müssen, ergibt sich dadurch, dass gegebenenfalls nicht festgestellt werden kann, ob ein Ereignis stattgefunden hat. So könnte z.B. innerhalb dieser 300ms-Periode ein Taster betätigt worden sein, der jedoch von der Software nicht registriert würde, da diese den Bus mit der Suche nach neuen Geräten besetzt. Mit diesem Argument sollte die Software den Bus nur nach einem festzulegenden Intervall von einigen Sekunden scannen. Dadurch erhöht sich die Wahrscheinlichkeit dass ein Tastendruck festgestellt werden kann.

Ein weiterer Fall der berücksichtigt werden muss, ist der der Offline-Knoten die nicht mehr mit dem Bus verbunden sind. Das Ansprechen der Knoten zum Ermitteln ihrer Zustände kann zu Fehlern führen. Bevor Knoten angesprochen werden, sollte sichergestellt werden, dass diese auch tatsächlich mit dem Bus verbunden sind. Eine entsprechende Routine benötigt nur einige Millisekunden und kann somit vor jedem Zugriff ausgeführt werden.

Ein weiterer Aspekt ist der, dass Knoten nur dann Daten übermitteln können, wenn diese angesprochen werden. Somit kann ein Knoten den Master nicht selbstständig über ein Ereignis informieren. Im Beispiel eines Tastendrucks würde ein Master den Tastendruck nicht registrieren, es sei denn er würde gerade zu diesem Moment diesen Knoten abfragen. Dies führt dazu, dass die Software so schnell und so oft wie möglich alle angeschlossenen Knoten abfragen muss. Dieses Problem des aktiven Pollens wäre im Produktiveinsatz sicher nicht vertretbar. Zur Vermeidung des aktiven Pollens ist eine andere Bus-Technologie sicher sinnvoller, jedoch reicht der 1W-Bus für den Demonstrator dieser Diplomarbeit aus.

Weitere Details zu diesen Abläufen wird im Kapitel des Software-Entwurfs behandelt.

Protokoll zwischen Geräte-Controller und Peer-Applikation

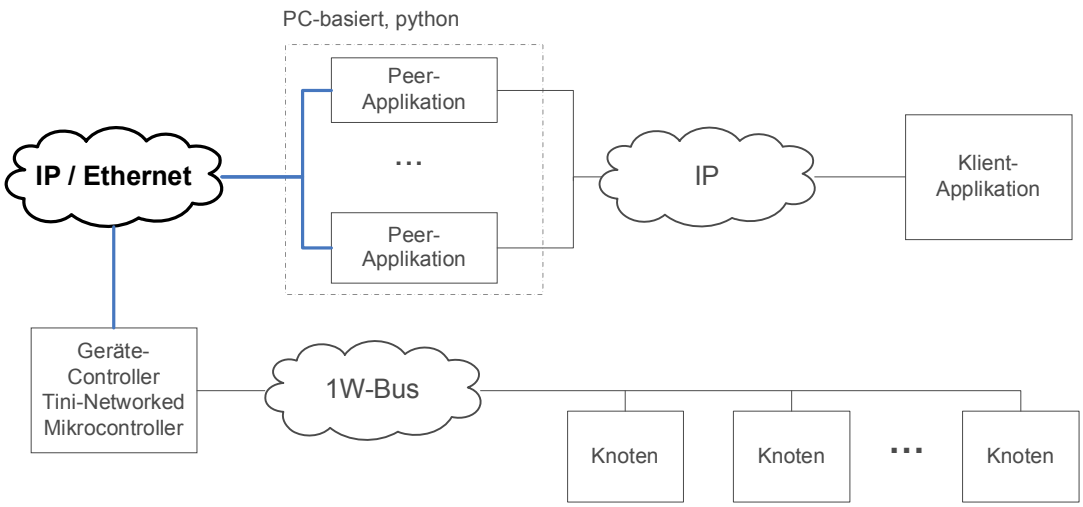


Abbildung 11 Kommunikation in der Gesamtstruktur

In diesem Teilkapitel soll auf den Entwurf eines Protokolls für die Kommunikation zwischen Geräte-Controller und Peer eingegangen werden, siehe Abbildung 11. Zuvor sollte jedoch motiviert werden, wieso ein eigenes Protokoll sinnvoll ist.

Die Wahl der Technologie zwischen Peer und Geräte-Controller fiel auf Ethernet mit TCP/IP. Da die Peers logische Verbindungen mit dem Geräte-Controller eingehen, liegt TCP als Kommunikationsprotokoll auf der Hand. Bedingt durch einige Schwachstellen in TCP, die noch etwas vertieft werden, sollte sich für ein alternatives Protokoll entschieden werden.

Potentielle Schwachstellen in TCP sind beispielsweise durch Havrilla [CERT1] nachgewiesen worden. Diese Schwachstellen sind protokollbedingt und unabhängig von der Implementierung. Wie in [CERT1] gezeigt wurde, können dritte Parteien TCP-Pakete in die Kommunikation einschleusen. Somit könnten Dritte, Einfluss auf die Knoten nehmen, so z.B. das Öffnen oder Schließen von Schaltern.

Eine weitere Art Angriffe gegen TCP zu starten, sind so genannte DoS-Angriffe. Hierbei ist das Ziel, den Server unerreichbar zu machen. Ein solcher Angriff ist z.B. "TCP Syn flooding". Hierbei wird der Server mit TCP-Syn-Paketen geflutet. Für jedes SYN-Paket muss der Server auf seinem TCP-Stack entsprechende Einträge vorsehen. Sind keine Ressourcen mehr verfügbar, ist der Server vorübergehend nicht mehr über das Netz erreichbar.

Beide Beispiele sollten illustrieren, dass TCP prinzipiell nicht geeignet ist, um die gewünschte Ausfallsicherheit zu erreichen. Eine Lösung stellt z.B. STCP dar [INET1]. STCP bereinigt die beiden erwähnten Probleme sowie einige andere. STCP ist leider nicht verbreitet und lässt sich in diesem Fall nicht einsetzen. Anders als STCP ist SCTP [IETF] bereits auf diversen Plattformen wie z.B. Linux oder FreeBsd implementiert. SCTP löst ebenfalls einige der Schwachstellen von TCP und berücksichtigt noch weitere Aspekte wie z.B. Multihoming, die allerdings in diesem Rahmen keine Rolle spielen.

Die Implementierung eines eigenen Protokolls scheint somit unausweichlich. Basierend auf UDP soll ein sicheres Kommunikationsprotokoll implementiert werden. Die Ansprüche an dieses Protokoll sollen im Folgenden motiviert werden.

Anforderungen an das Kommunikationsprotokoll

Die bereits erwähnten Probleme, die mit TCP einhergehen, sollten in dem neuen Protokoll berücksichtigt und beseitigt sein. Da die Peers sich in einer logischen Verbindung mit dem Geräte-Controller befinden, sollte nach dem Handshake die Garantie der Auslieferung der Pakete vom Protokoll gewährleistet werden. Ebenfalls sollte eine Sequenzierung der Pakete vorgenommen werden. Die Natur der Applikation impliziert, dass nur kleine Nachrichtenpakete zur Kommunikation verwendet werden. Dies führt dazu, dass eine Segmentierung seitens des Protokolls nicht erforderlich ist. Da die Kommunikation zwischen Geräte-Controller und Peer gesichert werden soll, wird eine Authentifizierung beider Seiten nötig. Zur Reduktion der Komplexität, soll ein geteiltes Geheimnis als ausreichend angesehen werden.

Nachfolgend alle Protokollkriterien noch einmal aufgelistet:

- Verwendung von UDP als Ersatzdatenträger, an Stelle von TCP
- Sequenzierung der Pakete durch das neue Protokoll
- Segmentierung nicht nötig
- Geteiltes Geheimnisprinzip
- Paketzustellgarantie wird vom Protokoll übernommen
- Flusskontrolle nicht notwendig

Ausgehend von einem Protokoll, entworfen von Henrici [ICSY1], lassen sich die eben erwähnten Kriterien implementieren. Das von Henrici entwickelte Protokoll basiert wie gewünscht auf UDP. Im Folgenden soll näher auf das Protokollschema von Henrici eingegangen werden.

Bei dem erwähnten Protokoll handelt es sich um ein 4-Wege Handshake mit Sicherung über Hashfunktionen. Beide Parteien verfügen über den geheimen Schlüssel. Die Partei die eine Verbindung aufbauen möchte, sendet ein INIT-Paket an den Server. Dieses gesendete INIT-Paket enthält eine Zufallszahl. Der Server erhält dieses Paket und sendet ein CHALLENGE-Paket zurück, in dem ebenfalls eine Zufallszahl enthalten ist. Der Hashwert über diese beiden Zufallszahlen und das geteilte Geheimnis bildet den Session-Cookie. Um dem Server zu beweisen, dass er als Klient einen Anspruch auf Verbindung hat, muss der Klient nun ein

ACK-Paket mit dem Hashwert über beide Zahlen zurücksenden. Dieser Session-Cookie muss ab nun mit jedem Paket mitgesendet werden. Der Server verwirft jedes Paket, das keinen korrekten Session-Cookie enthält. Ein Außenstehender ist nun nicht in der Lage, mit dem Server zu kommunizieren, sofern er nicht über einen gültigen Session-Cookie verfügt. Um den korrekten Session-Cookie zu generieren, benötigt der Klient das korrekte Passwort. Um jedoch zu vermeiden, dass der aktuelle Session-Cookie über das Abfangen von Paketen erfahren werden könnte, wird noch eine weitere Sicherheitsstufe mit eingeplant. Jedes Paket das nicht ein INIT- oder CHALLENGE-Paket ist, muss einen Hashwert über den Session-Cookie, die Sequenznummer, die Daten und das geheime Passwort mit jedem Paket zwecks Integritätssicherung mitsenden. Somit ist es nicht möglich die Pakete von außen zu manipulieren. Eine Änderung des Inhalts des Paketes würde der Server an dem nicht passenden Hashwert bemerken. Weiterhin wird gefordert, dass nur Pakete in aufsteigender Reihenfolge akzeptiert werden. Das Kopieren eines Pakets durch Dritte und das anschließende Wiedereinspeisen wird durch den Server bemerkt, da die Sequenznummer nicht die Erwartete wäre, dies schützt vor so genannten Replay-Angriffen.

Dieses Protokoll soll nun um die weiteren Anforderungen erweitert werden. Hauptschwierigkeit stellt hier die Zustellgarantie dar. Um die Komplexität an den Protokoll-Stack nicht unnötig zu erhöhen, wird die Zustellungsgarantie in vereinfachter Form implementiert. Im folgenden Abschnitt werden die genauen Anforderungen an die Zustellgarantie diskutiert.

Ausgehend von folgenden Überlegungen soll das Protokoll nun erweitert werden. Jedes gesendete Paket wird vorerst in einer Warteschlange gespeichert. Jedes empfangene Datenpaket muss durch ein ACK-Paket bestätigt werden. Ist ein Paket bestätigt, wird es aus dem Speicher des Senders endgültig gelöscht. Nach einer festen Wartefrist wird jedes Paket im Ausgangsspeicher wieder gesendet. Nach fünf erfolglosen Zustellungsversuchen wird dies als Verbindungsabbruch gewertet. Bei Empfang von Paketen deren Sequenznummer kleiner als die Erwartete ist, werden keine NAK-Pakete gesendet, sondern die Pakete werden umgehend verworfen. Die Pakete mit zu hoher Sequenznummer werden zwischengespeichert, bis die Lücken gefüllt werden können. Dieses vereinfachte Protokoll reduziert die Ansprüche an den Protokoll-Stack. Komplexe Verhaltensweisen wie bei TCP könnten leicht zu Fehlern

im Protokoll-Stack führen. Da keine großen Durchsätze erwartet werden, genügt dieses Protokoll den Ansprüchen vollkommen.

Da es sich beim Geräte-Controller um einen Mikrocontroller mit relativ beschränkter Speicherausstattung handelt, wird an den Geräte-Controller die Anforderung der Sequenzierung von Paketen nicht aufrecht erhalten. Die Sequenzierung würde einen zusätzlichen Speicherbedarf bedeuten, dem durch die knappen Ressourcen nicht Rechnung getragen werden kann. Somit werden Pakete die nicht in der gesendeten Reihenfolge empfangen werden, nicht durch den Mikrocontroller geordnet. Dies wird nicht zu Problemen führen, da Pakete an den Geräte-Controller atomare Kommandos enthalten die nicht über mehrere Pakete geteilt werden können.

Unter Berücksichtigung der festgelegten Anforderungen ergibt sich die in Abbildung 12 angegebene Paketstruktur.

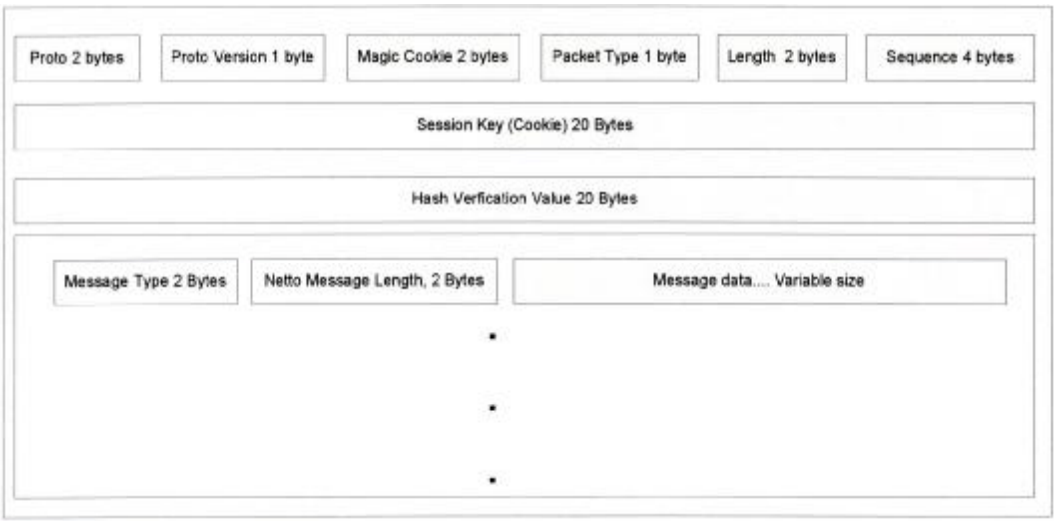

Proto 2 bytes
Proto Version 1 byte
Magic Cookie 2 bytes
Packet Type 1 byte
Length 2 bytes
Sequence 4 bytes
Session Key (Cookie) 20 Bytes
Hash Verfication Value 20 Bytes
Message Type 2 Bytes
Netto Message Length, 2 Bytes
Message data.... Variable size

Abbildung 12 Paketstruktur

Datenaustausch zwischen Geräte-Controller und Peer

In diesem Teil soll kurz auf die benötigten Konzepte für die Kommunikation auf Applikationsniveau eingegangen werden. Das vorgestellte Protokoll arbeitet bis zur Schicht 5 des ISO/OSI-Modells. Da Pakete als Kommunikationsschnittstelle unhandlich für die Applikation sind, wäre ein abstraktere Kommunikationsform wünschenswert. Die Einführung einer weiteren Schicht, die diese Umsetzung durchführt, ist wünschenswert.

Die Kommunikation, die zwischen Geräte-Controller und Peer stattfindet, ist prinzipiell nur der Austausch von kleineren Nachrichten. Eine Liste von Beispielen soll einen kleinen Eindruck vermitteln:

Einige Beispiel-Nachrichten vom Geräte-Controller zum Peer

- Schalter Y wurde gedrückt zum Zeitpunkt X
- Spannung an Eingang Z ist K Volt
- ID-Chip mit Seriennummer XXXXXX wurde am Bus angemeldet
- ID-Chip mit Seriennummer XXXXXX wurde am Bus abgemeldet
- Durchschnittliche Temperatur XX Grad Celsius

Einige Beispiel-Nachrichten vom Peer zum Geräte-Controller

- Setze Sample-Intervall von Sensor X auf Z Sekunden
- Schicke mir dein Error-Log-File
- Benutze neue Netzwerkkonfiguration X

Diese Beispiele verdeutlichen, dass es sich bei der Kommunikation zwischen Geräte-Controller und Peer-Applikation um kleine Nachrichten handelt. Somit wäre eine Schicht, die Nachrichten versendet und empfängt, sinnvoll. Diese Schicht sollte auch von der Paketschicht abstrahieren. Für den Nutzer dieser Schicht sollte uninteressant sein, wann oder wie Pakete gesendet werden. Dies führt zu einer vereinfachten Form der Kommunikation für die Applikation, da diese Schicht von Paketen abstrahiert und auch von den Empfängern, siehe Abbildung 13. Beim Senden von Nachrichten sorgt diese Schicht dafür, dass Nachrichten gesammelt und gemeinsam in einem Paket versendet werden.

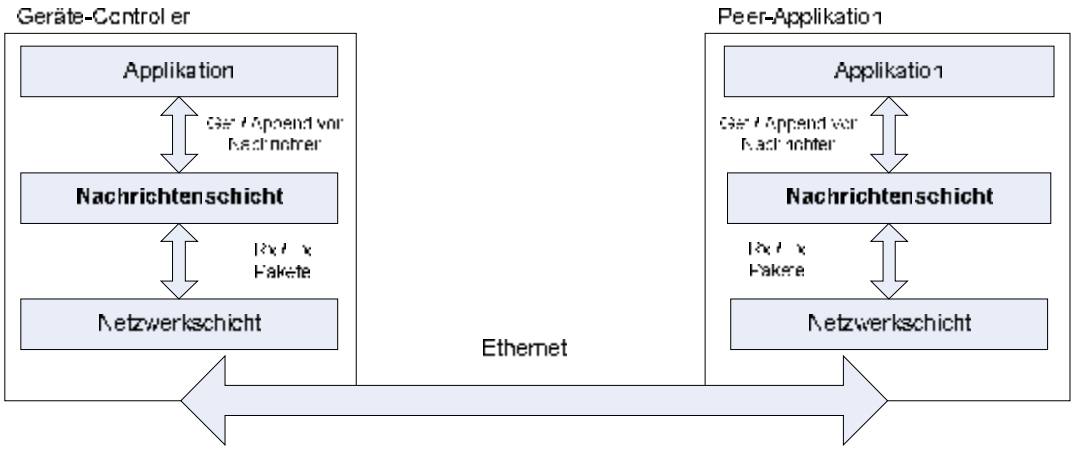


Abbildung 13 Nachrichtenschicht

Dieses Konzept vereinfacht die Kommunikation zwischen Peer und Geräte-Controller wesentlich. Allerdings existieren einige Fälle, in denen dem Nutzer die Sendung einer Nachricht als dringlich erscheint. Aus diesem Grund, werden zusätzlich zu dieser Schicht noch virtuelle Kanäle eingeführt. Der Nutzer kann beim Senden einer Nachricht angeben, welcher Kanal genutzt werden soll. Vorgesehen, wären einige Kanäle, gestuft nach Prioritäten. Der Kanal mit der höchsten Priorität würde beispielsweise das Senden eines Pakets sofort erzwingen, dadurch entstünde keine Verzögerung. Dies wäre wichtig bei eiligen Meldungen wie z.B. beim Druck eines Tasters. Für alle Kanäle, die nicht als eilig eingestuft sind, sollte man ein automatisches Flush vorsehen. Dieses Flush, würde den Puffer der Nachrichten nach einer gewissen Zeit automatisch absenden. Somit wird verhindert, dass es dazu kommen kann, dass Nachrichten über längere Zeiträume hinweg auf dem Geräte-Controller verweilen. Hinzu kommt noch der erzwungene Flush, dieser würde dann gestartet, wenn ein Überlauf des Nachrichtenpuffers anstehen würde.

Zusammenfassung

In den letzten Abschnitten wurden die Gründe für die Einführung eines eigenen Protokolls dargelegt. Um die Ausfallsicherheit gegenüber DoS-Angriffen zu erhöhen, wurde auf TCP zugunsten eines eigenen UDP-basierten Protokolls verzichtet. Dieses neue Protokoll besitzt ein 4-Way-Handshake und sichert die Kommunikation über Hashwertbildung ab. Die Hashwerte werden aus den Daten des Pakets und eines geteilten Geheimnisses berechnet. Zwecks Identifikation der logischen Verbindung benutzen beide Teilnehmer einen Session-Cookie, der in der Phase des Handshakes ermittelt wurde. Dieses Protokoll ist gegenüber Spoofing und "Packet injection" durch Dritte geschützt und erhöht somit die Sicherheit im

Vergleich zu TCP. Jedoch muss für Sequenzierung und garantierte Zustellung selbst gesorgt werden.

Um auf dem Applikationsniveau eine bequemere Schnittstelle zur Verfügung zu haben, wird eine weitere Schicht eingeführt. Diese Schicht hat zur Aufgabe, von den Datenpaketen hin zu Nachrichten zu abstrahieren. Die Applikation ist nun unabhängig von Datenpaketen und deren Größe. Dies hat den Vorteil, dass die Schicht automatisch entscheiden kann, ob genug Nachrichten angesammelt wurden, damit das Versenden eines Pakets sich lohnt.

Peer-Applikation und Klient-Applikation

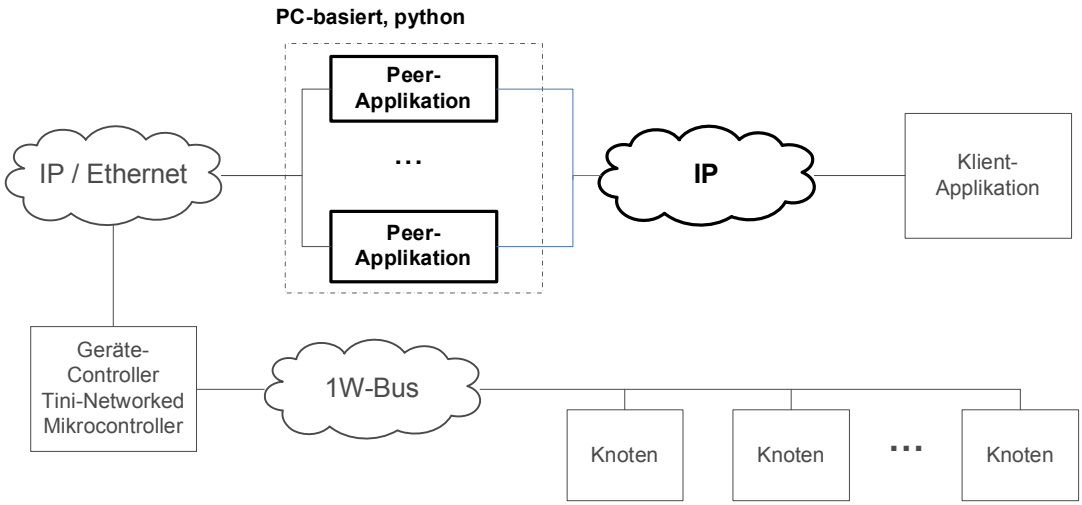


Abbildung 14 Peer-Applikation in der Gesamtstruktur

Der Peer erhält laufend Informationen seitens des Geräte-Controllers. Diese Informationen werden in der Informationsbasis des Peers gespeichert. Damit ein Klient auf diese Informationsbasis zugreifen kann, bedarf es einer entsprechenden Schnittstelle. In den Architekturentscheidungen wurde sich bereits dafür ausgesprochen, REST als Service-Architektur zu verwenden. In diesem Teil der Diplomarbeit wird ermittelt, wie dieses REST-Prinzip konkret als Schnittstelle angeboten werden kann.

Schnittstellenauswahl

Sowohl der Peer als auch der Klient sind PC-basierte Systeme, dadurch sind auch komplizierte Protokolle verwendbar. Da HTTP ein REST-basiertes Protokoll ist, steht der Schluss nahe, HTTP zu nutzen. Zwar unterstützt HTTP einfache REST-Aktionen, allerdings fehlt bei HTTP die Möglichkeit, andere Aktionen auf URIs anzuwenden als die, die in der Spezifikation stehen. Dies führt zu der Überlegung, dass HTTP lediglich als Transportmedium genutzt wird. Die eigentliche REST-Aktion wird in einer XML-Datei gespeichert. Diese XML-Datei wird dann mittels eines HTTP-Aufrufs an die Empfängerstelle übermittelt. Wenn der Klient nun eine Änderung an einem Knoten durchführen will, wird die Klient-Applikation eine XML-Datei erstellen, in der spezifiziert ist, welcher Knoten auf welche Weise manipuliert werden soll. Diese XML-Datei wird nun an die Peer-Applikation mittels HTTP übermittelt. Wenn nun die Peer-Applikation seinen Klienten ein Ereignis mitteilen will, wird dieser ebenfalls eine XML-Datei erstellen in der eine NOTIFY-Aktion spezifiziert ist. Diese XML-Datei wird dem Klienten dann per HTTP übermittelt. Dies bedeutet das beide Parteien über einen HTTP-Server verfügen müssen der diese Daten empfängt und auswertet. Es wird sich hierbei nicht um einen regulären HTTP-Server handeln, sondern nur um einen TCP-Socketserver, der die nötigsten Operationen beherrscht um die Anfragen verarbeiten zu können.

Da die Kommunikation zwischen Peer und Klient auf dem REST-Prinzip beruht, existiert keine aktive Verbindung zwischen beiden. Der Peer weiß prinzipiell nicht, dass Klienten verbunden sind und weiß nicht an welche Klienten ein Ereignis durchgereicht werden muss. Es ist somit nötig, dass der Klient in der Informationsbasis einträgt, an welchen Knoten er interessiert ist, dies wird im Folgenden als abonnieren bezeichnet. Der Klient kann somit alle oder spezielle Knoten abonnieren. Wenn nun für ein Knoten ein Ereignis eingetreten ist, so wird der Peer bei Bedarf ein NOTIFY an alle abonnierten Klienten senden. Der Peer findet diese Informationen in seiner Informationsbasis, die durch den Klienten hinzugefügt wurden.

Im folgenden Teil soll auf die Informationsbasis eingegangen werden. Diese Informationsbasis stellt eine Verbindung zwischen Knoten und Klienten her. Knoten werden in dieser Informationsbasis publiziert und werden so den Klienten zugänglich gemacht. Der Klient kann durch Nutzung der Informationsbasis, Knoten manipulieren.

Informationsbasis

In diesem Teil der Diplomarbeit wird die Datenstruktur der Informationsbasis besprochen. Die Peer-Applikation erhält laufend Nachrichten bezüglich der Knoten. Diese Nachrichten enthalten Informationen über eingetretene Ereignisse oder über gemessene Werte. Die Peer-Applikation benötigt eine entsprechende Datenstruktur, in der diese Informationen gespeichert werden können. Da diese Datenstruktur dazu genutzt wird, Informationen zu speichern, und gleichzeitig dazu genutzt wird, den Klienten Informationen daraus auszuhändigen, spielt die Informationsbasis eine zentrale Rolle, siehe Abbildung 15.

Bedingt durch die Architekturentscheidung, REST zu nutzen, sollte die Datenstruktur aus einer Menge von eindeutig adressierbaren Ressourcen bestehen. Angelehnt an das REST-Prinzip sollen diverse Aktionen auf den jeweiligen Ressourcen möglich sein. Durch Aktionen seitens des Klienten, lässt sich die Informationsbasis verändern.

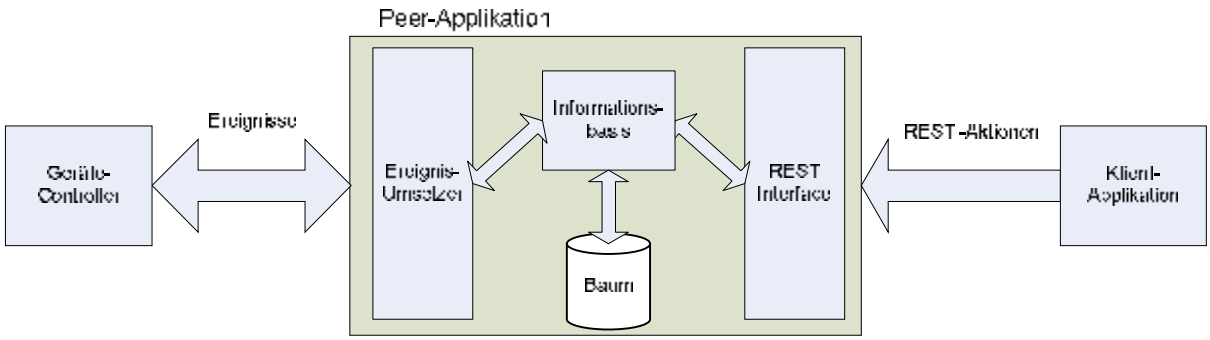


Abbildung 15 Informationsbasis

Die Rolle des Peers besteht nun darin, die binären Nachrichten des Geräte-Controllers auszuwerten und entsprechend in die noch zu definierende Informationsbasis einzutragen. Die Datenstruktur besteht bedingt durch das REST-Prinzip aus einer Menge von eindeutig adressierbaren Ressourcen. Es gilt nun zu definieren, wie Knoten als Ressourcen modelliert werden. Angelehnt an die MIB-Struktur von SNMP wäre eine baumartige Struktur denkbar. Die Blätter dieses Baumes würden dann Ressourcen entsprechen. Eine Ressource wäre z.B. der gemessene Temperaturwert eines Sensors. Im Folgenden ist ein Beispiel eines solchen Baumes angegeben:

/devices/offline/0BA5673A61511/present
/devices/offline/0BA5673A61511/lastsample
/devices/offline/0BA5673A61511/lastvoltage1
/devices/offline/0BA5673A61511/lastvoltage2
/devices/offline/0BA5673A61511/lastvoltage3
/devices/offline/0BA5673A61511/lastvoltage4
/devices/offline/EF04ABC323423/laststatechange
/devices/offline/EF04ABC323423/
/devices/offline/EF04ABC323423/pin1/level
/devices/offline/EF04ABC323423/pin1/state
/devices/offline/EF04ABC323423/pin2/level
/devices/offline/EF04ABC323423/pin2/state
/devices/offline/EF04ABC323423/pin3/level
/devices/offline/EF04ABC323423/pin3/state
/devices/offline/EF04ABC323423/pin4/level
/devices/offline/EF04ABC323423/pin4/state
/devices/online/0A9481EF61511/present
/devices/online/0A9481EF61511/lastsample
/devices/online/0A9481EF61511/lasttemperature
/devices/online/0A9481EF61511/01.02.2007-15:14:01
/devices/online/0A9481EF61511/01.02.2007-15:15:01
/devices/online/0A9481EF61511/01.02.2007-15:16:01
/devices/online/0A9481EF61511/01.02.2007-15:17:01
/devices/online/0A9481EF61511/01.02.2007-15:14:01

In diesem Beispiel sind drei Knoten vorhanden. Zum einen *0BA5673A61511*, der ein Quad-AD-Wandler Knoten darstellt, zum anderen *EF04ABC323423* ein E/A-Baustein und *0A9481EF61511* der ein Temperatursensor darstellt.

Diese Ressourcen können mittels GET-Aktion vom Klient abgefragt werden. Jedoch soll das Manipulieren dieser Ressourcen nur dann erlaubt sein, wenn dies im Sinne der Ressource ist, so sollen z.B. Messwerte nicht vom Klient veränderbar sein.

Der Zugriff auf die Resource */devices/offline/EF04ABC323423/pin3/state* mittels GET, würde z.B. 1 oder 0 ausgeben, 1 entspräche logisch High und 0 entspräche logisch Low. Den Zustand dieses Schalters könnte man mittels POST auf diese Ressource verändern. Ein POST mit 1 als Datum würde somit den Ausgang auf logisch High umschalten.

Entwurf der Software

Im folgenden Kapitel wird der Entwurf der einzelnen Softwarekomponenten besprochen. In erster Linie wird sich an den ermittelten Kriterien orientiert. Weiterhin werden in diesem Kapitel lediglich die groben Strukturen der Softwarekomponenten angesprochen. Für tiefer gehende Erläuterungen sei hier auf das Entwicklerhandbuch im Anhang verwiesen.

Geräte-Controller Software

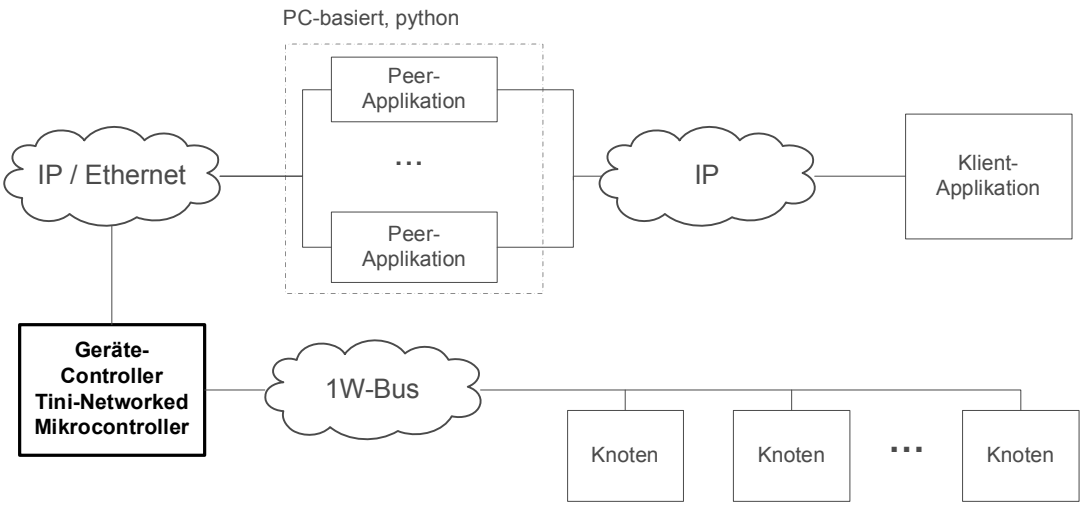


Abbildung 16 Geräte-Controller in der Gesamtstruktur

Der Geräte-Controller spielt eine zentrale Rolle im gesamten Rahmenwerk, siehe Abbildung 16. Im Kapitel der Technologiewahl wurde für den Geräte-Controller der Tini-Networked Mikrocontroller von Maxim-IC ausgewählt. Die Wahl der Programmiersprache fiel aus genannten Gründen auf C. Neben diesen Entscheidungen wurden auch einige Kriterien an die Software selbst gestellt. So sollen statische Speicherzuteilungen den dynamischen vorgezogen werden. Weiterhin muss die Software so konzeptioniert sein, dass der 1W-Bus so oft wie möglich abgefragt werden kann. Aktives Warten ist somit nicht erwünscht. Um aktives Warten zu vermeiden, müssen alle E/A im "non-blocking" Modus betrieben werden.

Da der KEIL Compiler der hier zur Verwendung kommt, nur C unterstützt, sind objektorientierte Entwurfsmethoden nicht möglich. Um dennoch ein gewisses Maß an Modularität zu gewährleisten, werden unterschiedliche Aufgabengebiete in abgeschlossenen Modulen gekapselt. Diese Module sollten in einen Hauptkern eingefügt werden. Idealerweise sollte die einzige Schnittstelle zwischen dem Kern und den Modulen, eine einzige Funktion sein. Module können so per Funktionszeiger in den Kern eingebunden werden. Der Kern kann so die Module bei Bedarf aufrufen. Den Funktionsmodus des Moduls kann per Parameterübergabe bestimmt werden. Wichtige Funktionsmodi wären z.B. "Initialisieren des Moduls" oder "Modul durchlaufen".

Wichtige Softwarekomponenten sind:

- Kern
- Netzwerk-Modul
- Peer-Modul
- Stream-Modul
- Persistentes-Logging-Modul
- 1W-Modul
- Globale modulunabhängige Funktionen
- Kapseln von Mikrocontroller abhängigen Funktionen

Der **Kern** bildet das Rahmenwerk, in dem alle globalen Funktionen zusammengefasst werden und bietet gleichzeitig auch die Schnittstelle zum Einfügrn von Modulen an.

Das **Netzwerk-Modul** wird unter anderen, folgende Aufgaben übernehmen:

- Initialisieren des Netzwerks (Binden an IP-Adresse,..)
- Empfang von Pakten übernehmen
- Prüfen der Korrektheit der einkommenden Pakete
- Transparentes Abwickeln der Handshakes
- Gesendete Pakete bis zum ACK zwischenspeichern
- Methoden anbieten, um Pakete zu generieren und zu versenden
- Die automatische Quittierung von empfangenen Paketen
- Das Neukonfigurieren des Netzes durch ein SETUP-Paket

Das **Peer-Modul** dient dazu, alle aktiven logischen Verbindungen zu speichern. Dieses Modul enthält eine Datenstruktur, in der alle Daten zu den jeweiligen Peers gespeichert werden. Diese Datenstruktur enthält unter anderen: Session-Cookies, versendete nicht quittierte Pakete, die Anzahl der fehlgeschlagenen Zustellungsversuche von Paketen, IP-Adresse und andere Daten. Im Folgenden werden die Aufgaben des Peer-Moduls aufgelistet:

- Das Speichern aller relevanten Informationen bezüglich der Peers
- Das Ermitteln von Peers die nicht mehr Verbunden sind
- Das Speichern von gesendeten Paketen, die noch nicht quittiert wurden

Das **Stream-Modul** setzt auf der Netzwerkschicht auf. Das Modul nimmt Pakete entgegen und fügt enthaltene Nachrichten in eine Nachrichtenwarteschlange ein. Gleichzeitig nimmt dieses Modul Nachrichten der Applikation entgegen und schickt diese über die Netzwerkschicht in Paketen zu den Peers. Im Folgenden sind alle Aufgaben dieser Schicht aufgelistet:

- Entgegennahme von Paketen über das Netzwerk-Modul
- Zwischenspeichern einkommender Nachrichten für spätere Auswertung
- Zwischenspeichern zu sendender Nachrichten für spätere Versendung
- Realisieren von Kanälen mit unterschiedlichen Sendeprioritäten
- Automatisches flushen der zu sendenden Nachrichten

Das **"Persistent Log"-Modul** dient zum Speichern von Laufzeitfehlern. Es stellt sicher, dass auch nach Ausschalten und späterem wieder Einschalten des Mikrocontrollers der Fehlerspeicher noch vorhanden ist. Das Modul greift auf das Dateisystem des Tinis zu und speichert alle Fehler in Dateien. Diese Dateien befinden sich im RAM des Tinis und werden durch die Batterie auch bei ausgeschaltetem Tini weiter erhalten. Im Folgenden, sind noch mal alle Aufgaben aufgelistet:

- Speichern von Fehlernachrichten im Tini-Speicher
- Garantieren der Erhaltung des Fehlerpuffers auch bei Absturz oder Abschalten des Mikrocontrollers
- Versendung des Fehlerspeichers bei Bedarf zum Peer

Das **1W-Modul** kapselt alle nötigen Funktionen, um den 1W-Bus zu konfigurieren und zu verwalten. Weiterhin wird dieses Modul eine Schnittstelle für Untermodule anbieten, in denen 1W-Geräte verschiedenen Typs eingebunden werden können. Dieses Modul wird ähnlich wie der Kern eine minimale Schnittstelle für Geräte-Module anbieten. Diese Geräte-Module beinhalten alle Routinen, die nötig sind, um mit einem speziellem 1W-Gerät umgehen zu können. Im Folgenden sind noch mal alle Funktionen aufgelistet:

- Initialisieren des 1W-Busses
- Scannen des Busses nach neuen Geräten
- Aufspüren von Offline-Geräten
- Das Versenden und Empfangen von Nachrichten bezogen auf 1W-Geräte
- Das Verwalten von nicht unterstützten Geräten
- Schnittstelle für Untermodule anbieten
- Den Untermodulen regelmäßig den Prozessor zuteilen
- Notifikationen über Online- oder Offline-Status von Geräten an alle Peers
- Notifikation von Ereignissen an alle Peers

Neben der ANSI-C-Kompatibilität des Source-Codes ist das Kapseln von Tini spezifischen Funktionen wesentlich, um die Portabilität auf andere Mikrocontroller zu unterstützen. Für weitere Informationen bezüglich der Realisierung der Software für den Geräte-Controller wird auf den Anhang "Geräte-Controller Software" verwiesen, weiterhin werden tiefere Einblicke im Benutzer- und Entwicklerhandbuch angeboten.

Peer-Applikation

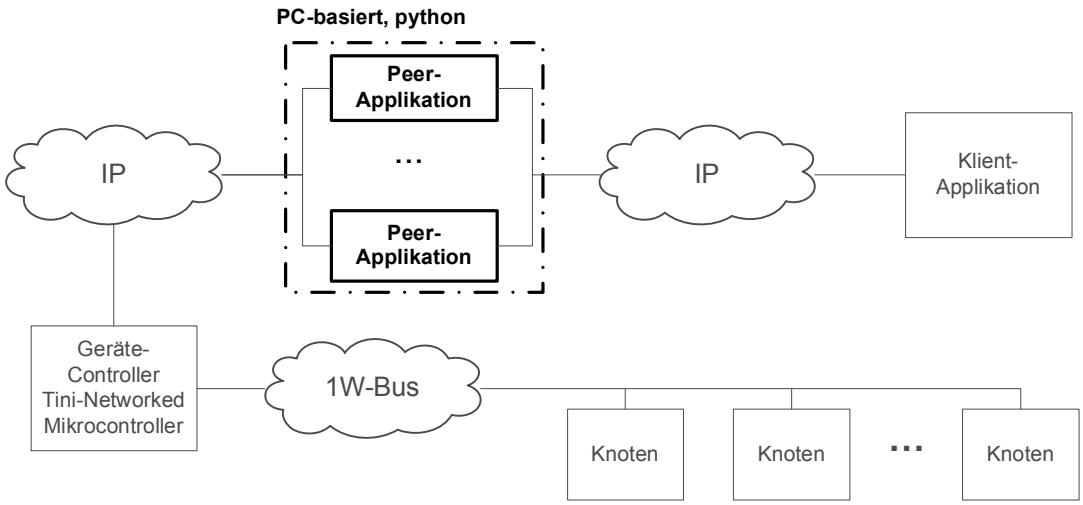


Abbildung 17 Peer-Applikation in der Gesamtstruktur

Da der Peer-Applikation eine zentrale Rolle zukommt, soll in diesem Teil der Entwurf etwas ausführlicher dargelegt werden, siehe Abbildung 17. Die Peer-Applikation bildet die Brücke zwischen den Daten der Knoten und den Klienten, die diese Daten nutzen wollen.

Wie bereits erwähnt, bildet die Informationsbasis das Rückgrad der Peer-Applikation. Jegliche Daten der Knoten werden in diesen Baum eingetragen. Mit der eindeutigen Adresse, die ein Zweig oder Blatt dieses Baumes besitzt, kann der Klient diese Informationen lesen oder manipulieren. Bevor auf die Modellierung dieses Baumes in Python eingegangen wird, werden die ermittelten Modulgruppen aufgelistet. Es ist sinnvoll, die Peer-Applikation in folgende Module aufzuteilen:

- Netzwerk-Modul
- Protokoll-Modul
- Stream-Modul
- DASM-Modul
- Log-Modul
- MIB-Modul
- MAP-Modul
- biREST-Modul

Das **Netzwerk-Modul** besitzt ähnliche Funktionalitäten wie das Netzwerk-Modul beim Geräte-Controller. Dieses Modul sorgt dafür, dass das Programm Pakete vom Netz lesen und schreiben kann. Unter anderem wird dieses Modul auch den vereinfachten Web-Server starten, der die Anfragen der Klienten beantwortet. Eine weitere Rolle dieses Moduls ist die des Ordnens der Pakete nach ihrer Sequenznummer.

Das **Protokoll-Modul** setzt auf die Netzwerkschicht auf und verwaltet alle Aktionen in Verbindung mit dem definierten Kommunikationsprotokoll. Hierzu gehören der Verbindungsaufbau und das Quittieren von Paketen.

Das **Nachrichten-Modul** abstrahiert auch hier die Netzwerkschicht und bietet der Applikation einen Strom von eingehenden und ausgehenden Nachrichten an. Ähnlich wie beim Geräte-Controller hat dieses Modul auch die Aufgabe, Nachrichten zu akkumulieren, um diese dann im richtigen Zeitpunkt zu senden.

Das **DASM-Modul** dient dazu, einkommende Pakete zu überprüfen und diese in entsprechende Objekte umzuwandeln. Jedes mögliche Paket besitzt eine entsprechende Klasse. Das Disassembler-Modul versucht die Pakete den entsprechenden Klassen zu zuweisen. Hierbei werden auch Überprüfungen über den Sinn der Pakete gemacht. "Zero length" Pakete z.B. werden verworfen. Das Vorgehen des DASM ist im Anhang unter "Paket-Logik der Peer-Applikation" genauer spezifiziert. Dieses Modul ermöglicht das einfache Erweitern der Kommunikation zwischen Geräte-Controller und Peer. Bei Einführung eines neuen Pakettyps muss lediglich das Format dieses Paketes im Programmcode angegeben werden. Änderung an den Kontrollstrukturen sind somit nicht nötig.

Das **LOG-Modul** bietet analog zum LOG-Modul des Geräte-Controllers Funktionen zum Ausgeben von Nachrichten oder Debug-Meldungen. Für weitere Informationen sei auf die Source-Codes zu diesem Modul verwiesen.

Das **MIB-Modul** verwaltet die Informationsbasis. MIB steht hier für "Management Information Base". Diese baumartige Struktur wird technisch mit assoziativen Arrays realisiert. Jeder Zweig ist selbst wieder ein assoziatives Array. Die Blätter dieses Baumes

stellen Funktionszeiger dar. Bei Anfragen an eines dieser Blätter von seiten des Klienten wird diese Funktion mit den REST-Parametern aufgerufen. Diese REST-Parameter beinhalten die Aktion (GET, DELETE..), die URI und optionale Daten. Mit diesen Parametern kann die aufzurufende Funktion genau ermitteln, was nun zu tun verbleibt. Dies hat folgende Vorteile: Wenn ein Gerät z.B. ein Temperatursensor auf dem Bus entdeckt wird, so kann die MIB eine Klasse eines Temperatursensors instanziieren und die Daten dieses Sensors in der MIB veröffentlichen. Die Veröffentlichung geschieht, indem den neuen Blättern, Methoden dieses Objektes zugewiesen werden. Anfragen an diese Blätter werden dann konkret in dem zugehörigen Objekt verarbeitet. Da auch andere Objekte in den Baum hinein geladen werden können, stellt dies eine besonders generische Methode dar. So können auch später neue 1W-Geräte nur durch die Erstellung einer Klasse unterstützt werden.

Das **MAP-Modul** dient dazu, die baumartige Struktur der Informationsbasis in eine andere abzubilden. Dies dient dazu, den Klienten eine besondere Sicht auf den Baum anbieten zu können. Die Original-Sicht auf den Baum ist vor allem technisch und somit für praktische Anwendungen zu unbequem, so sind z.B. l-W-Geräte mit ihren Seriennummern abgebildet. Durch das Map-Modul wird es ermöglicht, URI-Adressen in andere abzubilden.

Beispiel:
/device/0AFFAA0043/online

Eine GET Aktion auf diese URI würde 1 oder 0 zurückgeben. Da diese Adresse für Klienten nicht besonders aussagekräftig ist, kann man diese in eine andere abbilden, z.B. von "*/device/0AFFAA0043/online"* zu *"/tuerschloss/online"*. Dies würde dem Klienten den Knoten unter der Adresse *"/tuerschloss/online"* zugänglich machen.

Das **biRest-Modul** ist eine Besonderheit, die bisher noch nicht angesprochen wurde. Es ist möglich den Peer in zwei Teile aufzuspalten. Verdeutlicht wird die Notwendigkeit dieses Moduls im folgenden Szenario: Seien das Sensornetz, der Geräte-Controller und die Peer-Applikation hinter einem NAT-Router befindlich. Nun wäre es interessant, sich vom Internet aus mit der Peer-Applikation verbinden zu können. Da Anfragen vom Internet aus nicht in das NAT-Netzwerk durchdringen können, kann ein Klient keine Anfragen an die Peer-

Applikation richten. Mit diesem biRest-Modul kann man nun ein Teil der Peer-Applikation innerhalb des NAT-Netzwerks starten und den zweiten Teil außerhalb des NAT-Netzwerks starten. Der erste Teil der Peer-Applikation der sich im NAT-Netzwerk befindet, wird sich mit dem sich außerhalb befindlichen Peer-Teil verbinden. Es besteht nun eine logische Verbindung zwischen dem Peer im NAT-Netzwerk und dem Peer-Teil der sich außerhalb befindet. Die Klient-Applikation wird nun Anfragen an den Teil der Peer-Applikation richten der sich außerhalb des NAT-Netzwerks befindet. Dieses Teil wird die Anfragen nun durch die NAT-Hürde hindurch an die Peer-Applikation im NAT-Netzwerk durchreichen. Das biRest-Modul ermöglicht es der Klient-Applikation, REST-Aktionen durch die NAT-Hürde hindurch an die Peer-Applikation durchzureichen. So lassen sich auch außerhalb des LAN-Netzwerks die Knoten nutzen.

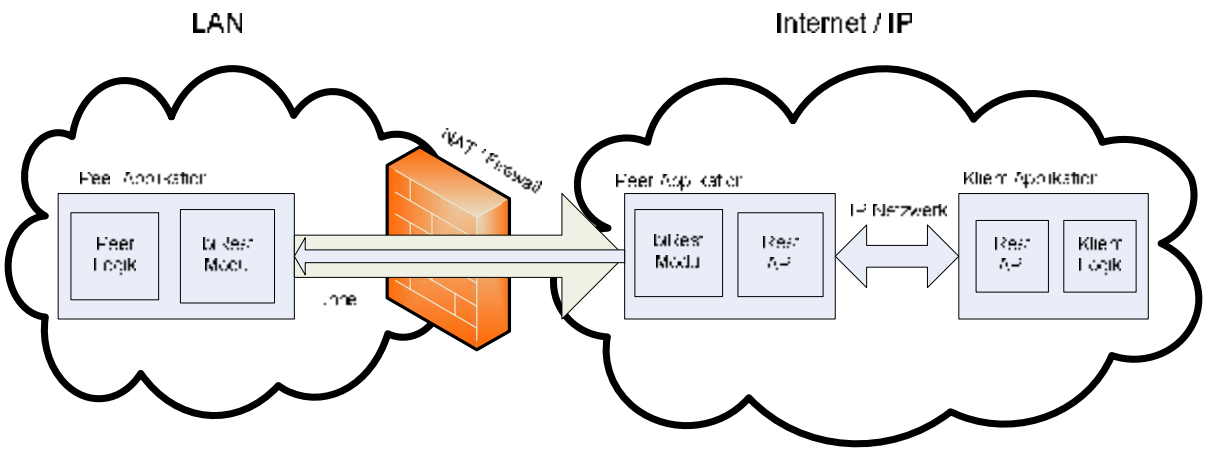


Abbildung 18 BiRest

Für weitere technische Details über die Peer-Applikation sei an das Entwicklerhandbuch im Anhang oder an den Anhang "Peer Software" verwiesen. Auch die Source-Codes sind eine gute Informationsquelle, diese sind auch in Form von Doxygen-Dokumenten verfügbar.

Einsatz

In diesem Kapitel soll die Funktionsfähigkeit mittels eines praktischen Fallbeispiels nachgewiesen werden. Hierzu wird ein Beispielszenario konstruiert, das mittels eines realen Demonstratoraufbaus geprüft werden soll.

Szenario

Ausgehend von der Idee, eine automatisierte Zugangsschleuse für ein Rechenzentrum zu entwerfen, wird folgendes Szenario konstruiert.

Die elektronische Tür bleibt ständig verschlossen. Tritt jedoch eine befugte Person an die Tür, soll diese automatisch geöffnet und nach Eintritt wieder geschlossen werden. Durch die hohen Leistungen heutiger Rechner soll die Tür im Fall einer Temperaturüberschreitung, automatisch zwecks Lüftung geöffnet werden. Dies mag nicht besonders realistisch sein, wird dennoch zwecks Demonstration so gewünscht.

Dieses einfache Szenario lässt sich ohne weiteres mit dem SASI-Rahmenwerk realisieren. Es werden lediglich einfache elektronische Schaltungen benötigt, die das elektronische Schaltelement des Knotens auf eine Ausgangsleistung verstärken, die das Schalten eines Relais oder Stellwerks ermöglicht.

Der Zutritt einer befugten Person wird mittels ID-iButton von Maxim-IC realisiert. Diese ID-iButtons sind 1W-Geräte, die eine Seriennummer enthalten. Wird die richtige Seriennummer erkannt, so wird das Schaltelement zum Öffnen der Tür angesteuert. Auf dem aufgebauten Demonstrator ist das Öffnen oder Schließen der Tür durch das Ein- und Ausschalten einer grünen beziehungsweise roten 235V Lampe dargestellt.

Die Temperaturüberschreitung lässt sich einfach mit einem 1W-Temperatursensor realisieren. Ein regelmässiges Überprüfen der aktuellen Temperatur soll dafür sorgen, dass eine Überschreitung erkannt wird.

Aufbau

Der Aufbau des Demonstrators, der dieses Verhalten simulieren soll, spaltet sich in zwei Teile auf. Zum einen muss die elektronische Schaltung für dieses Szenario entwickelt werden und zum anderen muss die Klient-Applikation, die die Logik implementiert, entworfen werden. Auf die elektronische Schaltung soll in diesem Teil nicht eingegangen werden, sondern auf die Implementierung der Klient-Applikation.

Das SASI-Rahmenwerk stellt einem alle Hilfsmittel zur Verfügung, um diese Applikation zu realisieren. Da es sich bei diesem Szenario um Ereignisse handelt die direkter Behandlung bedürfen, muss die Klient-Applikation ständig mit dem Peer verbunden sein. Mittels der zur Verfügung gestellten API kann mit Python ein entsprechender Klient in einigen Zeilen realisiert werden.

Grundsätzlich reduziert sich die Klient-Logik auf das Warten von Ereignissen, um anschliessend eine Reaktion auf die Ereignisse einzuleiten. Die Reaktion in diesem Fall wäre das Öffnen oder Schließen eines Schaltelements.

```
#!/usr/bin/python
import sys, time, wclib

wclib.start('localhost', 82);
wclib.programm_lock.acquire();
client = wclib.getClient(<host>, <port>);
client.do(wclib.newRA('POST','/subscribeall',<host:port>));



def handleNotification(pAction):

   if pAction.uri=='/temperatur/infos/lasttemp':
      if  float(pAction.data) > 31.0:
         print "Too hot, opening doors";
         client.do(newRA('POST', '/tuerschloss/state', '1'));

   if pAction.uri=='/karlheinz/infos/present':
      if  pAction.data == '1':
         print "Authorized person detected, opening doors";
         client.do(newRA('POST', '/tuerschloss/state', '1'));

      if  pAction.data == '0':
         print "Authorized person gone, closing doors";
         client.do(newRA('POST', '/tuerschloss/state', '0'));



try:
   while wclib.vbailoutcode==None:
      wclib.programm_lock.wait(0.1);
except:
   wclib.programm_lock.release();
```

Abbildung 19 Klient-Source-Code

In Abbildung 19 ist das Programm angeführt, um den Demonstrator zu programmieren.
Das Programm ist in drei Abschnitte aufgeteilt:

Abschnitt 1:

Initialisieren der API für das SASI-Rahmenwerk. Dies beinhaltet das Verbinden mit SASI und dem Abonnieren aller Knoten. Von nun an werden alle Ereignisse an die Klient-Applikation weitergeleitet.

Abschnitt 2:

Die Funktion "handleNotification" empfängt alle Notifikationen seitens des Peers. Der Parameter "pAction" ist eine so genannte Rest-Aktion und enthält alle notwendigen Daten, um eine Ressource zu identifizieren und um zu spezifizieren, was dieser Ressource

zugestoßen ist. Weiterhin enthält diese Funktion einige Vergleiche, um zu überprüfen, um welche Ressource es sich handelt. Handelt es sich z.B. um die Ressource "/karlheinz/infos/present", so ist klar, dass dieses Ereignis in Verbindung mit einem ID-iButton steht und somit das Datenfeld darüber Auskunft gibt, ob Karl Heinz sich angemeldet oder abgemeldet hat.

Wenn der Fall eintritt, dass Karl Heinz sich anmeldet: So wird eine REST-Aktion an den Peer gerichtet, die der Ressource **"/tuerschloss/state"** den Wert 1 zuordnet. Das SASI-Rahmenwerk setzt dieses um und teilt dem Geräte-Controller mit, dass der Zustand dieses Knotens geändert werden muss.

Abschnitt 3:

Im untersten Abschnitt wird der Main-Thread solange warten, bis ein Abbruch mittels CTRL-C erzwungen wird. Das Programm wird in dem Fall beendet

Versuch

Um den Erfolg der Konzepte und Methoden die in SASI zum Tragen kommen zu demonstrieren, müsste sich der Demonstrator gemäß dem erläuterten Szenario verhalten.

Wie erwartet, verhält sich der Demonstrator wie es das Szenario voraussetzt. Beim Einlegen des ID-iButton wird innerhalb weniger Augenblicke dieses Ereignis bis hin zur Klient-Applikation durchgereicht. Die Klient-Applikation wird alle nötigen Schritte unternehmen, um die Tür zu öffnen. Bei Entfernen des ID-iButton wird die Tür wie erwartet verschlossen.

Allerdings hat sich in diesem Versuchsaufbau auch eine Schwachstelle aufgezeigt. Das Scannen des Busses wird nur alle paar Sekunden durchgeführt. Dieses Intervall ist frei wählbar, sollte aber aus bereits behandelten Gründen nicht zu gering ausfallen. Durch dieses Intervall kann es durchaus länger dauern, bis der ID-iButton erkannt wird und das Schloss öffnen kann.

Eine weitere Schwierigkeit ergab sich manchmal durch das Einfügen und Entfernen des ID-iButton. Da das Einfügen durch menschliche Hand nicht sehr schnell durchgeführt wird, kam es in einigen Versuchen zur Störung des Busses. Dies ist nur insoweit tragisch, dass die bereits vorhandenen 1W-Geräte für Sekundenbruchteile vom Bus getrennt wurden. Dies wirkt sich jedoch nachteilhaft auf die Konfiguration der 1W-Geräte aus. Beim Anschließen eines Temperatursensors z.B. wird dieser vom Geräte-Controller auf eine gewisse Auflösung programmiert. Ist dieser nun gestört worden, kehrt dieser in seine Initialkonfiguration zurück. Fatal ist, dass der Geräte-Controller diese Zustandsänderung nicht wahrnimmt und somit künftig falsche Daten ausliest.

Die genannten Probleme, die sich im Versuch manifestiert haben, sind weniger konzeptioneller als mehr technischer Natur. Schaltungen, die das Entprellen beim Einfügen übernehmen, könnten dieses Problem beheben.

Zusammenfassung

Das Ziel dieser Diplomarbeit bestand darin, ein Rahmenwerk zu entwickeln, das auf kostengünstige und sichere Art und Weise, einfache Sensoren oder Aktuatoren mit großen offenen Netzen wie dem Internet verbindet. Im dem ersten Kapitel, in dem die Problemstellung klarer spezifiziert wurde, haben sich einige Kriterien ergeben, die ein sinnvolles Maß dafür sind, ob die Zielvorgaben erreicht wurden oder nicht. In folgenden soll evaluiert werden, inwieweit diese Kriterien erfüllt wurden. Zu Erinnerung, werden noch einmal die globalen Richtlinien aufgelistet.

- Hohe Ausfallsicherheit und Verfügbarkeit
- Geringe Kosten
- Generische Schnittstellen
- Flexibilität und Erweiterbarkeit

Das Kriterium der hohen Verfügbarkeit wurde mit diversen Methoden realisiert. So wurden z.B. ausgereifte Technologien, Redundanz, sichere Protokolle dazu verwendet um die Ausfallsicherheit zu maximieren. Das speziell für diesen Anwendungszweck entwickelte Protokoll benutzt Hashfunktionen um sicherzustellen, dass Pakete nicht durch Dritte manipuliert werden. Klassische Angriffsverfahren werden mit diesem Protokoll weitgehend verhindert.

Die Minimierung der Kosten wurde hauptsächlich durch die Überlegung umgesetzt, dass die Intelligenz der Knoten hin zu einer einzigen Komponente verlagert wird. Mit dieser Überlegung lässt sich auf jedem Knoten eine Ersparnis erzielen, die in der Summe beträchtlich ausfallen kann.

Mit der REST-Architektur wurde eine Wahl getroffen, die sich bereits in Protokollen wie dem HTTP bewiesen hat. Die Wahl einer baumartigen Informationsbasis als Datenstruktur ist sehr kompatibel zur REST-Architektur und ermöglicht eine ständige Erweiterbarkeit, die zugleich völlige Abwärtskompatibilität gewährleistet. Der Einsatz von HTTP als Kommunikationsmedium für die REST-Anfragen unterstützt zusätzlich die Zugänglichkeit dieses Services. Die Einspeisung einer REST-Anfrage in das Rahmenwerk wurde auf einfachste Weise durch das POSTen einer XML-Datei auf den Webserver der Peer-Applikation realisiert.

Die Flexibilität und einfache Erweiterbarkeit des SASI-Rahmenwerks wurde vor allem im Kapitel "Einsatz", zu Tage gebracht. Die Wahl von generischen Knoten erlaubt dem Entwickler alle möglichen Applikationen auf Schaltkreisniveau zu realisieren. Die Logik der Schaltkreise kann in der Klient-Applikation nach Belieben genutzt werden. Das SASI-Rahmenwerk agiert als Middleware und bedarf keiner Anpassung an die gewünschte Anwendung. Das SASI-Rahmenwerk ist somit offen für jegliche Anforderungen und bietet dem Entwickler eine bequeme Schnittstelle zu den Aktuatoren und Sensoren. Diese Schnittstelle vereinfacht dem Entwickler die Realisierung von Anwendungen.

Die im Kapitel über die Problemstellung motivierten Kriterien wurden alle erfolgreich umgesetzt, das SASI Rahmenwerk bildet eine kostengünstige Brücke zwischen Knoten und großen Netzen wie z.B.: dem Internet. Schlussendlich wurde der Erfolg der Konzepte und Methoden des SASI-Rahmenwerks anhand eines praxisnahen Fallbeispiels mittels eines Demonstrators aufgezeigt.

Ausblick

Aktuell wird in diesem Rahmenwerk die Logik der Applikation im Klienten implementiert. Das bedeutet, dass für den Betrieb einer Applikation die auf Ereignisse auf dem Bus reagiert, ein Klient ständig mit dem Rahmenwerk verbunden sein muss. Dies wurde explizit so modelliert, allerdings wäre eine Option denkbar, in der der Klient eine einfache Logik auf dem Geräte-Controller oder dem Peer platzieren könnte, nützlich. Diese einfache Logik könnte dann gleich auf dem Geräte-Controller oder Peer ausgeführt werden und somit würde die ständige Verbindung des Klienten nicht mehr benötigt. Hierzu wäre eine primitive Sprache Vorraussetzung, die auf dem Peer in Bytecode umgesetzt würde und dann Ereignisse seitens des Busses in Aktionen Seitens des Peers umsetzt.

In der Praxis hat sich gezeigt, dass das aktive Pollen des Knoten-Busses nachteilig ist. Bei einer großen Anzahl von Knoten, kann es dazu kommen, dass einige kritische Knoten wie z.B. Taster nicht schnell genug abgefragt werden um einen Tastendruck zu registrieren. Eine Lösung hierfür, wäre es einen anderen Bus zu wählen in dem die Knoten eigenständig Nachrichten an den Master senden können.

Anhang

Benutzer– und Entwicklerhandbuch

In diesem Teil des Dokumentes wird auf die Benutzung des entwickelten SASI-Rahmenwerks eingegangen werden. Das Benutzerhandbuch gliedert sich in eine kleine Einführung in SASI, gefolgt von einer kleinen Übersicht über die Architektur. Letztlich wird konkret auf die Aspekte der Installation und Konfiguration eingegangen. Im letzteren Teil dieses Handbuches wird auf Entwicklungsdetails eingegangen, die den Programmierer mit den nötigen Kenntnissen ausrüstet, um SASI weiter entwickeln zu können.

SASI

SASI steht für „Secure Attachement of Sensor devices to the Internet“ und hat zur Aufgabe, Knoten sicher an große Netzwerke wie z.B. dem Internet anzubinden.

Knoten haben als Eigenschaft, dass diese relativ wenig Intelligenz besitzen und somit ein Medium brauchen, mittels dem sie Daten zum Internet hin- und herschicken können. Bei SASI wird nicht die Idee verfolgt, Fähigkeiten in den einzelnen Knoten zu integrieren, sondern es wird versucht die Fähigkeiten auszulagern. Bei SASI werden diese Fähigkeiten zum Geräte-Controller und zu der Peer-Applikation ausgelagert.

Architektur

Konkret handelt es sich bei den verwendeten Knoten um 1W-Geräte von Maxim-IC. Diese Knoten sind einfache integrierte Bauelement die über einen einfachen Bus kommunizieren.

Der Geräte-Controller ist technisch gesehen ein Mikrocontroller, der über den Bus mit den Knoten kommuniziert und sich um die Administration des Busses kümmert. Dieser Geräte-Controller hat als weitere Aufgaben, die Knoten zu konfigurieren und nach außen hin eine Schnittstelle anzubieten. Da es sich beim Geräte-Controller um einen Mikrocontroller mit beschränkter Rechenleistung handelt, wird es vorgezogen, diesen nicht direkt mit dem Endbenutzer kommunizieren zu lassen. Es wird eine weitere Zwischenstufe hinzugenommen, die so genannte Peer-Applikation.

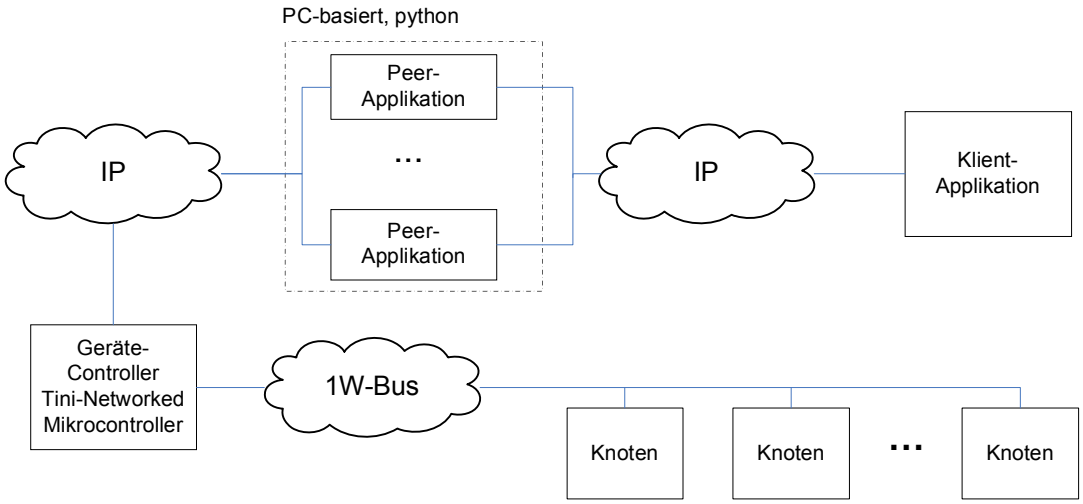


Abbildung 20 Gesamtarchitektur

Bei der Peer-Applikation handelt es sich um ein Stück Software, das in der höheren Programmiersprache Python realisiert wurde. Somit ist die Peer-Applikation höchst portabel und beschränkt den Benutzer nicht auf eine spezielle Plattform. Die Peer-Applikation dient dazu, die Daten des Geräte-Controller zu empfangen und weiter zu verarbeiten. Besonderes Augenmerk bei SASI liegt auf der sicheren Kommunikation zwischen Geräte-Controller und Peer-Applikation. Bei der Kommunikation zwischen Peer-Applikation und Geräte-Controller wird aus Sicherheitsgründen auf TCP verzichtet. Anstelle von TCP kommt ein eigens entwickeltes verbindungsorientiertes Protokoll auf UDP-Basis zum Einsatz. Nachdem der

Peer und der Geräte-Controller eine Verbindung etabliert haben, wird der Peer über alle verfügbaren Knoten informiert. Von dem Zeitpunkt an werden alle Ereignisse oder Daten der Knoten automatisch an den Peer übermittelt. Der Peer kennt somit immer den aktuellen Zustand der Geräte. Neben dem Empfang von Daten des Geräte-Controller kann der Peer auch die Konfigurationen der Knoten verändern. Die Peers, die zu dem Zeitpunkt verbunden sind, werden über die Konfigurationsänderungen in Kenntnis gesetzt. Weiterhin besteht die aktive Rolle des Peers darin, die empfangenen Daten aufzuarbeiten und diese in Form einer MIB zur Verfügung zu stellen. Die MIB steht hier für "Management Information Base" und ist an das Konzept der MIB von SNMP angelehnt.

Die MIB spielt eine zentrale Rolle im Peer. Diese MIB enthält alle Informationen der Knoten und bietet ebenfalls die Möglichkeit, die Konfigurationen der Geräte zu verändern. Da es sich bei der MIB nur um eine Datenstruktur handelt, wird ein Adapter benötigt. Dieser Adapter ist ein Teil der Peer-Applikation, die die MIB über diverse Schnittstellen der Außenwelt zur Verfügung stellt. Mit Außenwelt ist in diesem Kontext das Internet oder ein IP-Netzwerk gemeint.

Die MIB ist nicht die einzige Analogie zwischen der Peer-Applikation und SNMP. Das zugrunde liegende Kommunikationskonzept der Peer-Applikation basiert auf REST. Die MIB kann mittels GET, PUT, POST oder DELETE manipuliert werden. Diese REST-Anfragen sind an die Schnittstelle der Adapters zu richten.

Aktuell unterstützt dieser Adapter eine Web-Schnittstelle. Anfragen werden per HTTP an den Webserver des Adapters gerichtet. Die Anfrage muss eine XML-Datei als POST enthalten, die die Anfrage spezifiziert.

Somit lassen sich Anfragen von einem Klienten direkt an den Adapter richten. Um allerdings auch die Möglichkeit zu bieten, asynchrone Notifikationsmeldungen vom Peer zu erhalten, ist auch das umgekehrte Verhalten vorgesehen. Ein Klient kann alle oder spezielle Knoten abonnieren. Das Abonnieren bedeutet hier, dass der Klient über Ereignisse bezüglich des abonnierten Knotens informiert wird. Hierfür richtet der Peer eine HTTP-Anfrage an den Klienten. Diese Anfrage muss ebenfalls eine XML-Datei als POST enthalten. Auch hier dient die XML-Datei dazu, die REST-Anfrage zu präzisieren.

Installation und Konfiguration

Da es sich beim Geräte-Controller um einen Mikrocontroller handelt, wird ein spezieller Compiler für Mikrocontroller benötigt. Zum Entwickeln der Software wurde der Keil-Compiler verwendet. Die erzeugte Binärdatei muss mittels MTK2 von Maxim-IC über die serielle Schnittstelle auf den Mikrocontroller geladen werden. Der Mikrocontroller bootet nun selbstständig. Am Geräte-Controller sind nun keine weiteren Schritte notwendig. Die Konfiguration des Mikrocontrollers geschieht über die Peer-Applikation.

Bei der Peer-Applikation ist die Installation wesentlich einfacher. Es reicht, die gepackte Peer-Applikation zu entpacken und mittels geeigneter Python-Installation zu starten. Es werden keine speziellen Python-Module oder Pakete benötigt.

Um die Peer-Applikation zu starten, reicht folgendes Kommando:

host:> python peer.py

Prinzipiell werden von der Peer-Applikation Root-Rechte benötigt. Dies hängt damit zusammen, dass der Adapter sich an Port 80 bindet. Dies lässt sich jedoch umgehen, indem man die Portbindungen ändert, dazu später mehr.
Es wird empfohlen, sich mit den Kommandozeilenargumenten der Peer-Applikation vertraut zu machen. Dies lässt sich mit folgendem Kommando durchführen:

host:> python peer.py --help

Wenn die Peer-Applikation immer mit den gleichen Argumenten gestartet wird, ist zu empfehlen, die „sasi.conf“ Datei einzustellen. Diese Datei enthält alle wesentlichen Konfigurationsoptionen, die von Peer-Applikation benötigt werden. Diese Datei wird bei jedem Start gelesen und als Konfigurationsgrundlage genommen. Mit den Kommandozeilenparametern kann man diese Konfigurationsoptionen überschreiben.

Bevor näher auf die Konfiguration der Peer-Applikation eingegangen wird, soll noch kurz die Konfiguration des Geräte-Controllers erwähnt werden. Wie erwähnt, geschieht die

Konfiguration des Geräte-Controller über den Peer. Falls der Geräte Controller noch nicht konfiguriert wurde, startet dieser mit einer Initialkonfiguration. Diese lautet wie folgt:

IP-Adresse: 192.168.2.30

Netzwerk: 255.255.255.0/24

Gateway: 0.0.0.0

Masterpasswort: 12345678

Shared Key: gauss

Nach dem erstmaligen Start des Geräte-Controller ist dieser also unter der genannten Adresse erreichbar. Diese Werte können nun mittels der Peer-Applikation angepasst werden. Mit folgendem Kommandozeilenaufruf wird ein interaktiver Setupmodus gestartet, der die nötigen Informationen vom Benutzer abfragt und am Ende an den Geräte-Controller übermittelt.

host:> python peer.py –x 12345678

Es sei angemerkt, dass das Masterpasswort nicht veränderbar ist, sondern fest in das Programm eincodiert ist. Weiterhin ist der Geräte-Controller nur innerhalb von 10 Minuten nach dem Start konfigurierbar. Sollte man nicht innerhalb dieses Zeitfensters eine Konfiguration vorgenommen haben, sollte man den Geräte-Controller einfach neu starten.

Nachdem nun der Geräte-Controller eingestellt wurde, sollte man die neuen Konfigurationsdaten in die *„sasi.conf“* Datei eintragen. Nach dem Start der Peer-Applikation wird diese sich sogleich mit dem Geräte-Controller verbinden. Nachdem die Verbindung erfolgreich war, wird die Peer-Applikation laufend Informationen erhalten. Diese Informationen sind über den Adapter verfügbar. Der Adapter ist ein Modul, das in der Peer-Applikation integriert ist. Es ist jedoch möglich, den Adapter auszulagern. Es ist möglich, die Peer-Applikation ohne Adapter zu starten und den Adapter im Standalone-Modus zu starten.

Die Peer-Applikation und der Adapter werden über ein bidirektionales REST miteinander kommunizieren. Dieses bidirektionale REST wurde hier so realisiert, dass es auch NAT durchqueren kann. Es ist somit möglich, die Peer-Applikation in einem NAT-Netzwerk zu betreiben und den Adapter auf einem fernen Rechner außerhalb des NAT-Netzwerks zu platzieren. Dem Adapter ist es möglich, direkt mit dem Peer zu kommunizieren, da Peer und Adapter eine Verbindung nach dem Start eingehen.

Folgende Kommandozeile startet die Peer-Applikation ohne Adapter:
host:> python peer.py –l

Folgende Kommandozeile startet lediglich den Adapter:
host:> python peer.py –w

Die Adresse an die sich der Adapter bindet, wird in der „sasi.conf“ Datei spezifiziert. Es ist nötig, den Adapter vor der Peer-Applikation zu starten, da sich die Peer-Applikation mit dem Adapter verbindet. Dies ist unbedingt notwendig da nur die Peer-Applikation vom NAT-Netzwerk heraus einen Kommunikationskanal etablieren kann, umgekehrt kann der Adapter sich nicht mit der Peer-Applikation verbinden, da Verbindungen am NAT-Gateway scheitern würden.

Die Peer-Applikation enthält im Verzeichnis „contrib/“ zwei Beispielprogramme. Zum einen den „webclient“. Hierbei handelt es sich um eine kleine Web-Applikation, die unter einer gewöhnlichen Apache–PHP4-Installation läuft. Diese Web-Applikation verbindet sich mit dem Adapter und ermöglicht es einem, die MIB zu durchforsten, Werte abzufragen oder zu ändern. Man sollte diese Applikation nutzen, um sich etwas mit der MIB vertraut zu machen. Der Nachteil dieser Web-Applikation liegt darin, dass keine NOTIFY-Meldungen des Peers empfangen werden können. Dies liegt in der Natur des HTTP-Protokolls. Die Web-Applikation besitzt nämlich keinen HTTP-Server an die die Peer-Applikation die NOTIFY-Meldungen ausliefern könnte.

Das zweite Programm „wanclient“ befindet sich ebenfalls im „contrib“ Verzeichnis. Dieses Programm verbindet sich dauerhaft mit dem Adapter. Im Gegensatz zur Web-Applikation, besitzt dieser Klient einen Web-Server der die NOTIFY-Meldungen des Peers entgegennehmen kann. Es erhält somit Notifikationen über Ereignisse der Knoten und kann entsprechend reagieren. Dieses Programm ist sehr klein und lässt sich einfach abändern, um die gewünschte Logik zu realisieren. Hiermit sind beliebig komplexe Verhalten möglich.

Schlussendlich soll noch kurz auf das Map-Modul eingegangen werden. Dieses Modul ist optional und dient dem Zweck, die MIB in eine andere Form zu transformieren. Die original MIB ist für manche Zwecke zu nahe an der Technik. So werden z.B. Knoten mit ihren Seriennummern aufgelistet.

Beispiel:

/raw/devices/switches/1AVB05165/[...]

Bei diesem Beispiel handelt es sich um ein Knoten, mit dem man schalten kann. Es ist in der Praxis sinnvoller, diesem Knoten einen Namen zu geben. Diesem Zweck dient das Map-Modul. Das Map-Modul wandelt also URIs um. So kann man die MIB ganz der Zweckmäßigkeit des Projektes anpassen. Die Mapping-Regeln werden in der Datei „map.conf“ angegeben. Die Datei enthält einige Kommentare, die die Regelsyntax verdeutlichen.

Weiterentwicklung

In diesem Teil wird auf die Weiterentwicklung des SASI Rahmenwerks eingegangen. Als erstes wird auf den Geräte-Controller und die Knoten eingegangen. Im weiteren Teil wird die Erweiterung der Peer-Applikation erläutert. Es ist ratsam, den ersten Teil dieses Kapitels zu lesen, um die grundlegende Architektur des SASI Rahmenwerks zu verstehen.

Knoten

Es gibt eine große Anzahl von 1W-Geräten die auf dem Markt zur Verfügung stehen. Jedes dieser 1W-Geräte unterstützt andere Funktionalitäten. SASI unterstützt zu diesem Zeitpunkt vier Knoten:

- Temperatursensor
- Eingabe und Ausgabe
- ID-Buttons
- AD-Wandler

Wenn ein weiteres Gerät unterstützt werden soll, so ist es nötig, den Programmcode vom Geräte-Controller und der Peer-Applikation anzupassen. Da die 1W-Geräte alle anders über den Bus angesteuert werden, ist es leider nicht möglich, eine generische Schnittstelle für alle Typen zu definieren. Prinzipiell kann man jedoch mit den vier unterstützten Geräten durch Schaltungen eine Vielzahl von Applikationen entwickeln. Da die Schaltung für den Geräte-Controller unwichtig ist, muss auch keine Erweiterung hierfür programmiert werden. Die Logik der Schaltung ist für SASI unerheblich, da SASI lediglich die Kommunikation zwischen den Knoten und dem Benutzerprogramm regelt.

Nichts desto trotz wurde vorgesehen, weitere 1W-Typen einzuprogrammieren. Es befindet sich im Geräte-Controller Source-Tree im Modul „onewire“ einen generellen 1W-Typus, der keine Funktionalität besitzt. Dieser Typ heißt *„m_onewire_skel“*, *„skel“* steht für Skeleton und soll bedeuten, dass es lediglich den Rahmen festsetzt. Die Funktionskörper müssen an den 1W-Typ angepasst werden.

Geräte-Controller

In diesem Abschnitt soll kurz erläutert werden, wie die Architektur des Geräte-Controller realisiert wurde. Da der verwendete Keil-Compiler kein OOP unterstützt, wurde das Programm klassisch funktional entworfen. Weiterhin wurde aus Robustheitsgründen auf Threading verzichtet. Es wurde auch versucht, die Anzahl der dynamischen Allokationen zu minimieren, um den Raum für Laufzeitprobleme zu verkleinern. Die meisten Datenstrukturen wurden statisch deklariert, sofern dies möglich war.

Grundsätzlich durchläuft das Geräte-Controller-Programm eine Endlosschleife. In dieser Schleife wird den so genannten Modulen der Prozessor zugeteilt. Jedes der Module besitzt eine Run-Funktion, die die nötigen Aufgaben dieses Moduls abarbeiten soll. Wichtig hierbei ist, dass diese Funktion nicht blockiert. Alle Vorgänge müssen so schnell wie möglich ohne Warten abgearbeitet werden.

Module sind prinzipiell unabhängig von einander. Im Programm existieren allerdings einige Vernetzungen, da verschiedene Netzwerkschichten in verschiedenen Modulen realisiert wurden. So hat z.B. die Netzwerkschicht ein Modul und die Stream-Schicht hat ein eigenes Modul, jedoch benutzen beide Funktionalitäten des Anderen.

Es existiert auch hier eine Art Skeleton-Modul. Bei Erweiterung des Programms wird angeraten, dieses Modul zu kopieren und anzupassen. Module werden mittels der Mount-Funktion in der „core.c“ eingefügt. Jedes Modul muss eine Dispatcher-Funktion implementieren, die dazu dient, dem Modul ein Kommando zu senden. So wird dieser Dispatcher-Funktion z.B. ein INIT-Kommando beim Start des Geräte-Controllers zugesandt. Den RUN-Aufruf wird ebenfalls über diese Dispatcher-Funktion realisiert.

Um die Funktionsweise der aktuell implementierten Module zu verstehen, sei auf die Dokumentation im Source-Code verwiesen.

Peer-Applikation

Die Peer-Applikation wurde mit objektorientierten Methoden entwickelt und vereinfacht somit dem Entwickler die Erweiterung des Rahmenwerks. Die Peer-Applikation verfügt über einige Module und Untermodule. Die wichtigsten Module seien hier kurz erwähnt:

- net
- birest
- mib
- daemon
- log

Das Netzwerkmodul kapselt alle netzwerktechnischen Funktionen, wie z.B. das Binden an den Socket, das Lesen der eingehenden Pakete und so weiter. Grundsätzlich werden Pakete in Objekte umgewandelt. Dies macht es nötig, dass für jeden Typ von Paket eine Definition dieses Typus existieren muss. Dies wird mittels des „dasm“ Moduls realisiert. Dieses Modul liest die Pakete und versucht, eine Klasse zu finden, die zu diesem Paket passt. Dies macht das Erweitern des Kommunikationsprotokolls erheblich einfacher. Es müssen keine hartcodierten Routinen geändert werden, sondern nur eine Typdefinition des Paketes angegeben und die Identifikationsnummer eingetragen werden.

Das biRest-Modul kapselt die Kommunikation zwischen Peer-Kern und dem Adapter. Mit diesem Modul können Peer-Kern und Peer-Adapter getrennt werden und sogar über NAT-Gateway oder Proxy-Server miteinander kommunizieren.

Das Daemon-Modul bildet den Sammelpunkt für alle Funktionalitäten und Module. Der Daemon liest die Konfigurationen und stellt alle anderen Module ein. Der Daemon ist sozusagen der Manager der Peer-Applikation.

Das Log-Modul bietet einfache Log-Funktionalitäten. So kann man beispielsweise wichtige Nachrichten speichern oder Debug-Nachrichten ausgeben. Der Verbosity-Level der Applikation entscheidet, welche der Nachrichten angezeigt werden. Mit dem „-v“ Parameter auf der Kommandozeile kann man die Debug-Nachrichten einschalten.

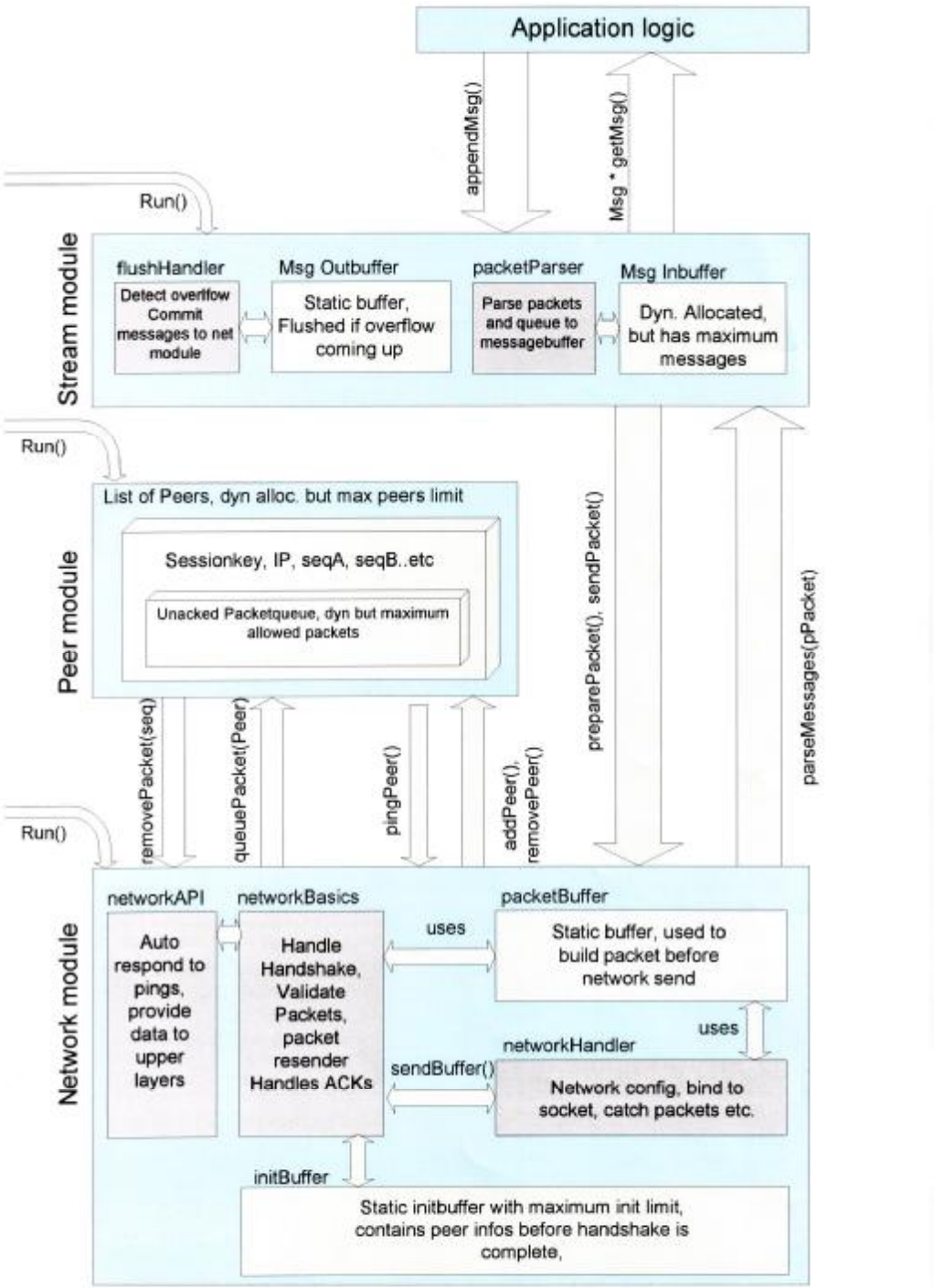
Das MIB-Modul kapselt die zentrale Datenstruktur. Es handelt sich dabei technisch gesehen um einen Baum der auf assoziativen Arrays basiert. Jedem Blatt in diesem Baum wird eine Funktion zugewiesen. Wird nun eine Anfrage an eine URI gemacht, so wird dieses Blatt im Baum gesucht und anschließend die dort gespeicherte Funktion aufgerufen. Die Funktion erhält sämtliche Informationen der Anforderung, z.B. GET , POST, DELETE etc. Wie die Funktion nun auf diese Anfragen reagiert, ist an den Sinn dieser Ressource gebunden und kann frei implementiert werden.

Wenn man einen neuen Knoten einfügen möchte, so sollte man diesen in Form einer Klasse modellieren. Es existiert die abstrakte „device“ Klasse, die Basisfunktionalität eines 1W-Gerätes beherbergt. Alle implementierten 1W-Geräte leiten sich von dieser Klasse ab. Man sollte sich beispielsweise die anderen, bereits vorhandenen 1W-Geräte ansehen, um ein besseres Verständnis bezüglich der Zusammenhänge zu erlangen.

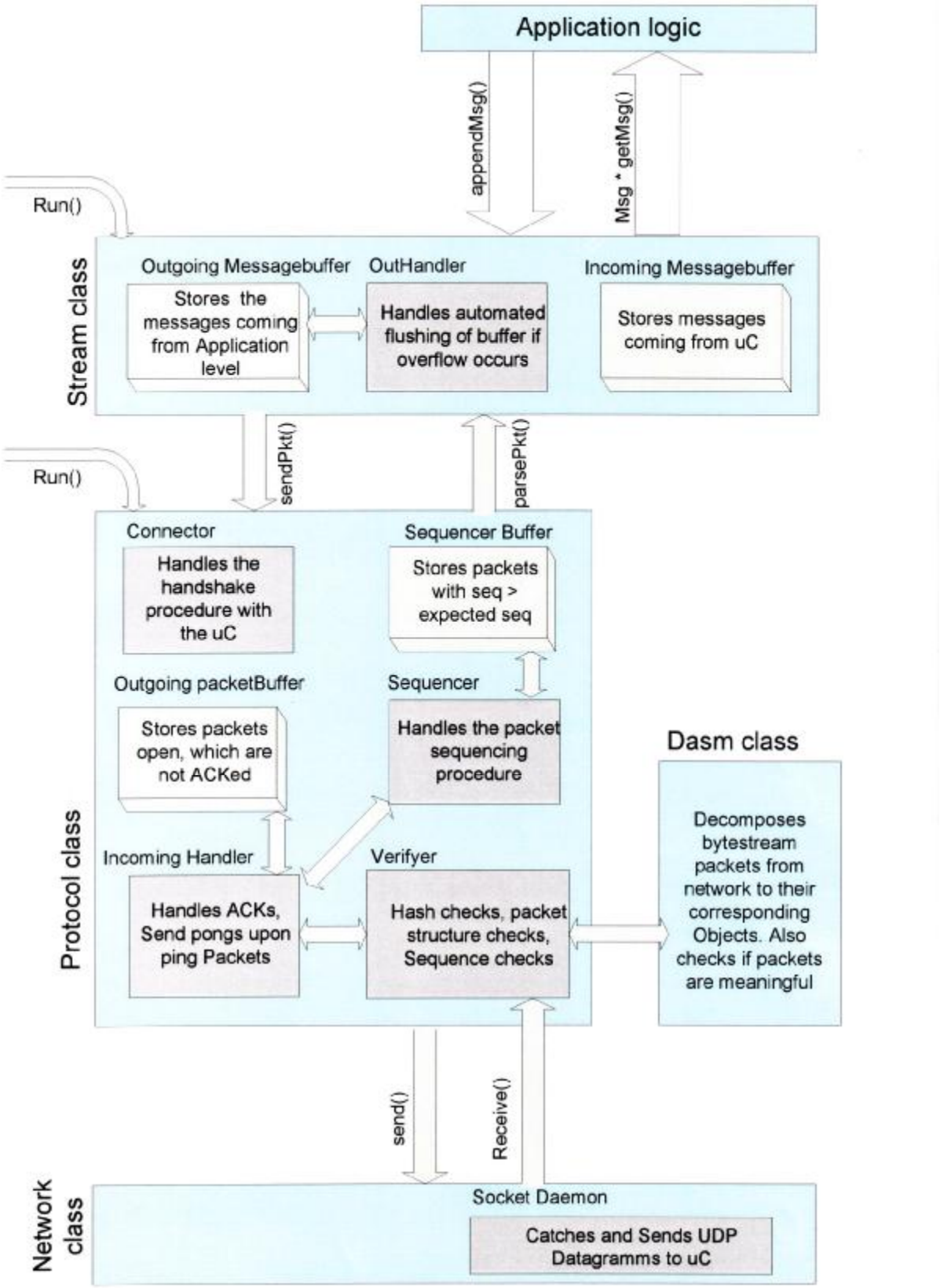
Genauere Spezifizierung der Softwarearchitektur und des Kommunikationsablaufs befinden sich im Anhang unter "Softwareentwürfe". Schlussendlich sei hier auf die Source-Codes der jeweiligen Programme verwiesen. Diese sind reichlich im Doxygen-Stil dokumentiert.

Softwareentwürfe

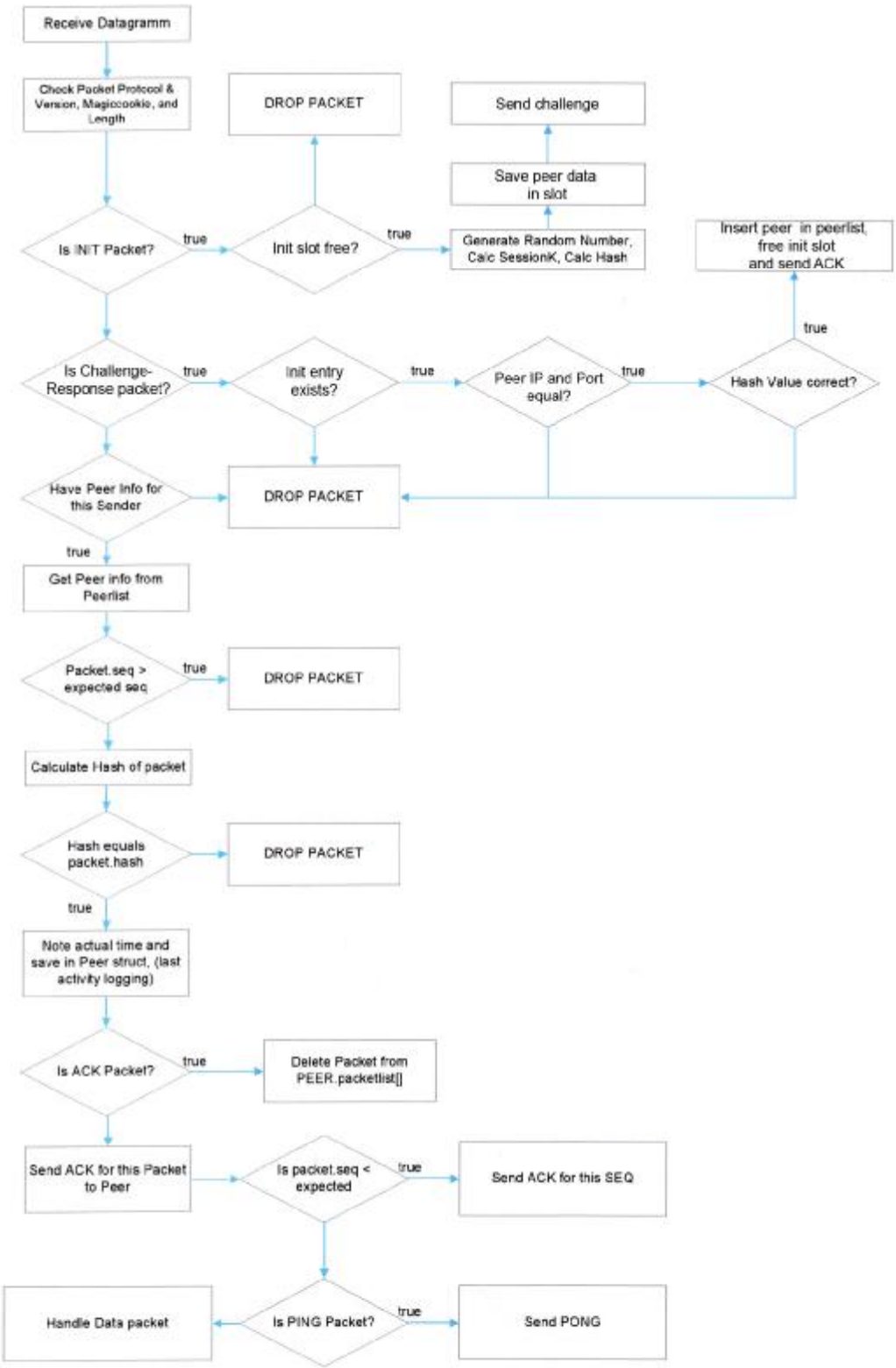
Geräte-Controller Software



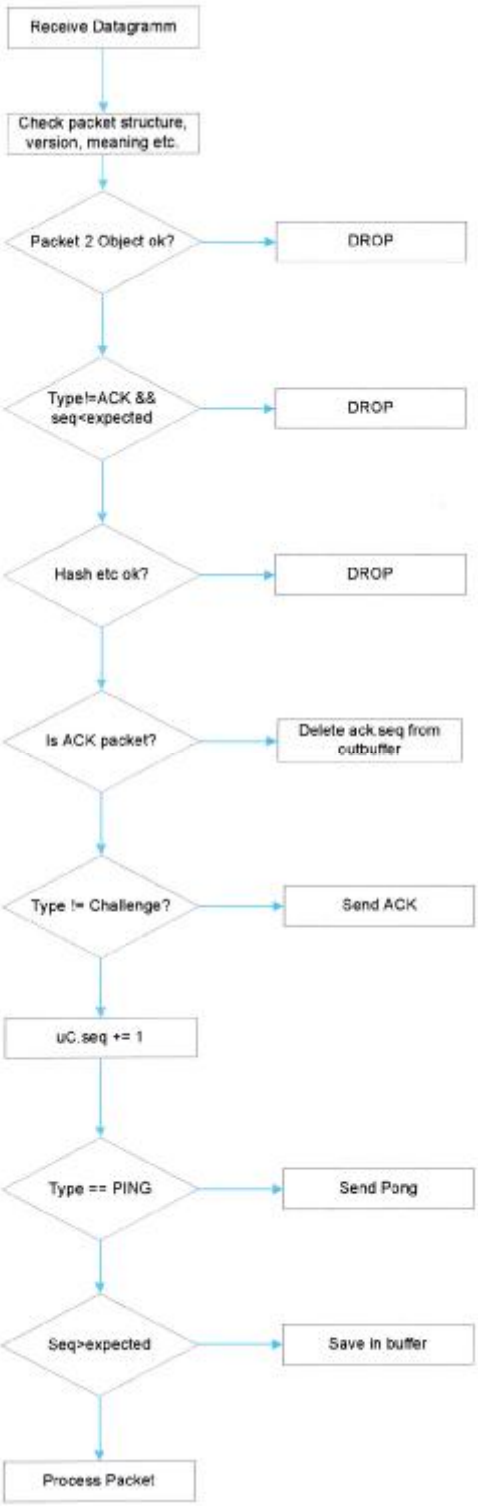
Peer Software



Paket-Logik des Geräte-Controller



Paket-Logik der Peer-Applikation



Literaturverzeichnis

[CERT1] Jeffrey S. Havrilla: "Multiple TCP/IP implementations may use statistically predictable initial sequence numbers", 2001 http://www.kb.cert.org/vuls/id/49844 11/05/2007

[IAB1] Kary Mayer, Wolfgang Fritsche: "IP-Enabled Wireless Sensor Networks and their integration into the Internet", International conference on Integrated Internet ad-hoc and sensor networks, ACM International Conference Proceeding Series Vol. 138, 2006

[ICITA1] L. Tan and K. Taylor: “Mobile SCADA with Thin Clients - A Web Demonstration”, Commonwealth Scientific and Industrial Research Organisation, 2002

[ICSY1] Dirk Henrici: "Bridging the Gap Between Pervasive Devices and Global Networks", in submission, 2007

[IETF] R. Stewart, Q. Xie, et al.: "Stream Control Transmission Protocol", 2000 http://tools.ietf.org/rfc/rfc2960.txt

[INET1] Tsutsumi Toshiyuki, Yamaguchi, Suguru: "Secure TCP -- Providing Security Functions in TCP Layer", 1995 http://www.isoc.org/HMP/PAPER/144/html/paper.html, 11/05/2007

[INSIGHT] Murat Demirbas, Ken Yian Chow, Chieh Shyan Wan: "Internet-Sensor Integration for Habitat Monitoring", Proceedings of the 2006 International Symposium on World of Wireless Mobile and Multimedia Networks, 2006

[LSIR] Karl Aberer, Manfred Hauswirth, Ali Salehi: "A Middleware For Fast And Flexible Sensor Network Deployment", Proceedings of the 32nd international conference on Very large data bases - Volume 32 pp: 1199-1202, 2006

[MAX1] Maxim-IC: "Add Control, Memory, Security, and Mixed-Signal Functions with a Single Contact", 2007 http://pdfserv.Maxim-IC.com/en/an/AN3989.pdf 07/05/07

[MIT1] Raffi Krikorian: "Bringing IP to the leaf node", O'Reilly Emerging Technology Conference, 2003
http://ttt.media.mit.edu/impact/i0oreilly.pdf 27/05/07

[SI1] A. Flammini, P. Ferrari, E. Sisinni, D. Marioli, A. Taroni: "Sensor interfaces: from field-bus to Ethernet and Internet ", Sensors and Actuators A: Physical, Volume 101, Number 1, 2002

[WEBSCADA] A. Ekpruke, Richard W. Wall: "Low-Cost Sensor System for Alarm Reporting" http://www.mrc.uidaho.edu/cisr/labs/comms/pubs/WEB%20SCADA.pdf 24/05/2007

[WEISER] Mark Weiser: "The Computer for the 21st Century", Scientific American Ubicomp Paper, 1991
http://www.cc.gatech.edu/fac/Gregory.Abowd/hci-resources/area-bok/papers/p94-weiser.pdf 24/05/2007

[WSN2006] Karl Aberer, Manfred Hauswirth, Ali Salehi: "Middleware support for the Internet of Things" Fachgespräch "Drahtlose Sensornetze", Universität Suttgart, 2006
ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/TR-2006-07/TR-2006-07.pdf 25/05/2007

Abbildungsverzeichnis

Abbildung 1 IP-fähige Knoten 12
Abbildung 2 Geräteanschluss über Gateway 13
Abbildung 3 Knoten und Geräte-Controller 23
Abbildung 4 Peer und Geräte-Controller 24
Abbildung 5 Architektur 27
Abbildung 6 Knoten in der Gesamtstruktur 33
Abbildung 7 1W-Master und -Slaves 36
Abbildung 8 Geräte-Controller in der Gesamtstruktur 36
Abbildung 9 Peer-Applikation in der Gesamtstruktur 38
Abbildung 10 1W-Bus in der Gesamtstruktur 41
Abbildung 11 Kommunikation in der Gesamtstruktur 42
Abbildung 12 Paketstruktur 46
Abbildung 13 Nachrichtenschicht 48
Abbildung 14 Peer-Applikation in der Gesamtstruktur 49
Abbildung 15 Informationsbasis 51
Abbildung 16 Geräte-Controller in der Gesamtstruktur 54
Abbildung 17 Peer-Applikation in der Gesamtstruktur 58
Abbildung 18 BiRest 61
Abbildung 19 Klient-Source-Code 64
Abbildung 20 Gesamtarchitektur 71

Abkürzungen

1W	Onewire Technology from Maxim-IC
ACK	Acknowledgement
ATM	Asynchronous Transfer Mode
CAN	Controller Area Network
DoS	Denial Of Service
ETH	Ethernet
GSN	Global Sensor Network
HMI	Human Machine Interface
HTTP	Hypertext Transfer Protocol
I0	Internet0
I2C	Inter-Integrated Circuit
IP	Internet Protocol
ISO	Internationale Organisation für Normung
JVM	Java Virtual Machine
KISS	Keep it Short and Simple
LAN	Local Area Network
MIB	Management Information Base
NAK	Negative Acknowledgement
NIC	Network Interface Card
OSI	Open Systems Interconnection
PC	Personal Computer
PLC	Programmable Logic Controller
RAM	Random Access Memory
REST	Representational State Transfer
RFC	Request for Comment
ROM	Read Only Memory
RPS	Remote Power Switch
RTU	Remote Terminal Unit
SASI	Secure Attachment of Sensors to the Internet
SCADA	Supervisory Control and Data Acquisition
SNMP	Simple Network Management Protocol
SRAM	Static Random Access Memory
STCP	Secure TCP

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
USB	Universal Serial Bus
WAN	Wide Area Network
WWW	World Wide Web
XML	Extensible Markup Language